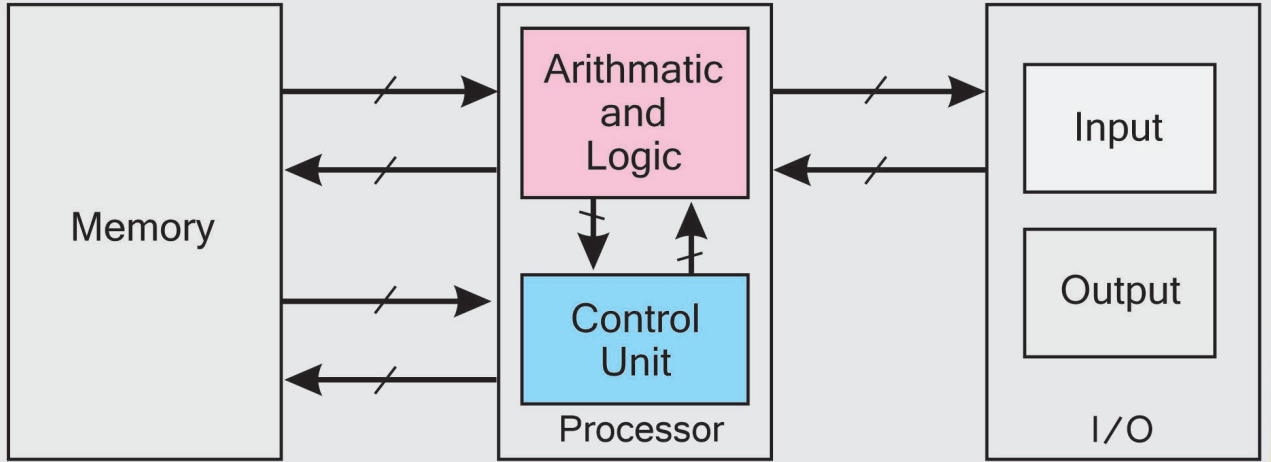


अखिल भारतीय तंत्रशिक्षण परिषद
All India Council for Technical Education



संगणक प्रणाली संघटन



डॉ. सोनल यादव

संगणक प्रणाली संघटन

लेखक

डॉ. सोनल यादव

प्राध्यापक,

संगणक शास्त्र अभियांत्रिकी नॅशनल इंस्टिट्यूट ऑफ टेक्नोलॉजी,
रायपूर छत्तीसगढ़ - 492010

अनुवादक

डॉ. श्रीकांत देवराव झाडे

सहयोगी प्राध्यापक

संगणक विज्ञान आणि अभियांत्रिकी
नागपूर इंस्टिट्यूट ऑफ टेक्नॉलॉजी, नागपूर

भाषा पुनरालोकनकर्ता

डॉ. प्रविण प्रकाश आडिवरेकर

सहयोगी प्राध्यापक,

संगणक विज्ञान आणि अभियांत्रिकी (डेटा सायन्स)
ए. पी. शाह इंस्टिट्यूट ऑफ टेक्नॉलॉजी,

All India Council for Technical Education

Nelson Mandela Marg, Vasant Kunj,

New Delhi, 110070

BOOK AUTHOR DETAILS

Dr. Sonal Yadav, Assistant Professor, Dept. of Computer Science & Engineering (CSE), National Institute of Technology Raipur, Chhattisgarh.

Email ID: syadav.cse@nitrr.ac.in

TRANSLATOR DETAILS

Dr. Shrikant Zade, Associate Professor, Computer Science and Engineering, Nagpur Institute of Technology, Nagpur.

Email ID: zadeshrikant@nit.edu.in

LANGUAGE REVIEWER DETAILS

Dr. Pravin P. Adivarekar, Associate Professor, Department of Computer Science & Engineering (Data Science), A.P. Shah Institute of Technology.

Email ID: ppadivarekar@apsit.edu.in

BOOK COORDINATOR (S) – Marathi Version

1. Dr. Sunil Luthra, Director, Training and Learning Bureau, All India Council for Technical Education (AICTE), New Delhi, India. **Email ID:** directortlb@aicte-india.org
2. Sanjoy Das, Assistant Director, Training and Learning Bureau, All India Council for Technical Education (AICTE), New Delhi, India. **Email ID:** ad2tlb@aicte-india.org
3. Reena Sharma, Hindi Officer, Training and Learning Bureau, All India Council for Technical Education (AICTE), New Delhi, India. **Email ID:** hindiofficer@aicte-india.org
4. Avdesh Kumar, JHT, Training and Learning Bureau, All India Council for Technical Education (AICTE), New Delhi, India. **Email ID:** avdeshkumar@aicte-india.org
5. Dr. Sanjay Laxmikant Nalbalwar, Professor, Department of Electronics and Telecommunication, Dr. Babasaheb Ambedkar Technological University, Lonere, Maharashtra, India. **Email ID:** nalbalwar_sanjayan@yahoo.com

March, 2025

© All India Council for Technical Education (AICTE)

ISBN : 978-93-6027-066-7

All rights reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from the All India Council for Technical Education (AICTE).

Further information about All India Council for Technical Education (AICTE) courses may be obtained from the Council Office at Nelson Mandela Marg, Vasant Kunj, New Delhi-110070.

Published by All India Council for Technical Education (AICTE), New Delhi.



Attribution-Non Commercial-Share Alike 4.0 International (CC BY-NC-SA 4.0)

Disclaimer: The website links provided by the author in this book are placed for informational, educational & reference purpose only. The Publisher do not endorse these website links or the views of the speaker / content of the said weblinks. In case of any dispute, all legal matters to be settled under Delhi Jurisdiction, only.



अखिल भारतीय तकनीकी शिक्षा परिषद्

(भारत सरकार का एक सांविधिक निकाय)

(शिक्षा मंत्रालय, भारत सरकार)

नेल्सन मंडेला मार्ग, वसंत कुंज, नई दिल्ली-110070

दूरभाष : 011-26131498

ई-मेल : chairman@aicte-india.org

ALL INDIA COUNCIL FOR TECHNICAL EDUCATION

(A STATUTORY BODY OF THE GOVT. OF INDIA)

(Ministry of Education, Govt. of India)

Nelson Mandela Marg, Vasant Kunj, New Delhi-110070

Phone : 011-26131498

E-mail : chairman@aicte-india.org

प्रो. टी. जी. सीताराम
अध्यक्ष
Prof. T. G. Sitharam
Chairman

अग्रलेख

अभियंते हा कोणत्याही आधुनिक समाजाचा कणा असतो. ते चमत्कारांसाठी तसेच जगभरातील जीवनाच्या सुधारित गुणवत्तेसाठी जबाबदार आहेत. अभियंत्यांनी अधिक विकसित आणि अभूतपूर्व पद्धतीने मानवतेला अधिक उंचीवर नेले आहे.

अखिल भारतीय तंत्रशिक्षण परिषदेने (AICTE) देशातील तंत्रशिक्षणाच्या बळकटीकरणासाठी कोणतेही प्रयत्न सोडले नाहीत. मानवजातीच्या सर्वांगीण कल्याणावर भर देणारे भारताला आधुनिक विकसित राष्ट्र बनवण्यासाठी AICTE नेहमीच दर्जेदार तांत्रिक शिक्षणाला प्रोत्साहन देण्यासाठी वचनबद्ध आहे.

गेल्या दशकात AICTE द्वारे अनेक उपक्रम घेतले गेले आहेत ज्यांना आता राष्ट्रीय शैक्षणिक धोरण (NEP) 2020 द्वारे गती दिली गेली आहे. भारताच्या माननीय पंतप्रधानांच्या दूरदर्शी नेतृत्वाखाली NEP च्या अंमलबजावणीमध्ये प्रादेशिक शिक्षणासाठी तरतूद करण्याची कल्पना आहे. त्यात सर्वांना प्रादेशिक भाषांमध्ये शिक्षण देण्याची तरतूद आहे. याद्वारे प्रत्येक पदवीधर पुरेसा सक्षम बनतो आणि नवोन्मेष आणि उद्योजकतेद्वारे राष्ट्रीय वाढ आणि विकासासाठी योगदान देऊ शकतो याची खात्री केली जाते.

एआयसीटीई गेल्या काही वर्षांपासून अथकपणे काम करत असलेल्या क्षेत्रांपैकी एक म्हणजे अंडर ग्रॅज्युएट आणि डिप्लोमा स्तरावर उच्च दर्जाची मूळ तांत्रिक सामग्री विविध भारतीय भाषांमधील नामवंत शिक्षकांनी तयार केलेली आणि अनुवादित केलेली आहे. त्यांच्या अभियांत्रिकी शिक्षणाच्या दुसऱ्या वर्षाला शिकणाऱ्या विद्यार्थ्यांसाठी, AICTE ने 87 पुस्तके ओळखली आहेत, जी मूळ सामग्रीच्या पूर्ण निर्मितीवर 12 भारतीय भाषांमध्ये अनुवादित केली जातात - हिंदी, तमिळ, गुजराती, ओडिया, बंगाली, कन्नड, उर्दू, पंजाबी, तेलुगू, मराठी, आसामी आणि मल्याळम. इंग्रजी माध्यमाव्यतिरिक्त, विविध भारतीय भाषांमधील पुस्तके विद्यार्थ्यांना त्यांच्या संबंधित मातृभाषेतील संकल्पना समजून घेण्यासाठी मदत करणार आहेत.

AICTE च्या वतीने, मी सर्व मान्यवर लेखक, समीक्षक आणि अनुवादक यांचे विक्रमी कालावधीत प्रशंसनीय योगदान दिल्याबद्दल उच्च प्रतिष्ठित संस्थांमधून त्यांचे मनापासून आभार व्यक्त करतो.

AICTE ला विश्वास आहे की या निकालावर आधारित मूळ सामग्री इच्छुकांना या विषयावर आकलन आणि अधिक सहजतेने प्रभुत्व मिळवण्यास मदत करेल.

टी.जी.सीताराम.

प्रा. टी. जी. सीताराम

(अध्यक्ष)

ऋणनिर्देश

अभियांत्रिकी आणि तंत्रज्ञानाच्या विद्यार्थ्यांसाठी तांत्रिक पुस्तक प्रकाशित करण्यासाठी ए.आय.सी.टी.ई. (AICTE) ने केलेल्या सूक्ष्म नियोजन आणि अंमलबजावणीबद्दल लेखक कृतज्ञ आहेत. तसेच या पुस्तकास विद्यार्थ्यांसाठी अनुकूल, सोयीचा आणि कलात्मक दृष्टीकोनातून उत्तम आकार देणारा बनवल्याबद्दल; समीक्षक प्रा. मीलन मेहता यांच्या अमूल्य योगदानासाठी मनापासून आभार व्यक्त करत आहेत. हे पुस्तक ए.आय.सी.टी.ई. चे सदस्य, विषय तज्ञ आणि लेखकांच्या विविध सूचनांचे फलित असून त्यांनी आपल्या देशातील अभियांत्रिकी शिक्षणाचा अधिक विकास करण्यासाठी त्यांचे मत आणि विचार व्यक्त केले आहेत. मी हे देखील मोठ्या सन्मानाने सांगतो की हे पुस्तक ए.आय.सी.टी.ई. च्या आदर्श अभ्यासक्रमाशी आणि राष्ट्रीय शैक्षणिक धोरण-२०२० (NEP-2020) च्या मार्गदर्शक तत्वांनुसार आहे. आपल्या देशातील प्रादेशिक भाषांमधील शिक्षणाला चालना देण्यासाठी हे पुस्तक (अनुसूचित) भारतीय प्रादेशिक भाषांमध्ये अनुवादित केले जात आहे. शैक्षणिक आणि इतर विविध क्षेत्रातील योगदानकर्ते आणि कार्यकर्ते ज्यांची प्रकाशित पुस्तके, पुनरावलोकन लेख, शोध निबंध, छायाचित्रे, टिपणे, संदर्भ आणि इतर मौल्यवान माहिती प्रकाशित केलेली आहे त्यांचे श्रेयनिर्देश व आभार व्यक्त करणे क्रमप्राप्त आहे. मला हे पुस्तक लिहिण्यास मदत करणारे, डॉ. सोनल यादव यांचे मी मनापासून आभार मानू इच्छितो. डॉ. यादव यांनी पुस्तक लिखाणाच्या सुरुवातीपासून माझ्यासोबत काम केले आहे आणि उदाहरणे, प्रभावली आणि प्रयोग लिहिण्यासाठी विशेष सूचना दिल्या.

डॉ. श्रीकांत देवराव झाडे, सहयोगी प्राध्यापक, संगणक अभियांत्रिकी, नागपूर इन्स्टिट्यूट ऑफ टेक्नोलॉजी, नागपूर यांच्या अनुवादासाठीच्या योगदानाबद्दल आणि डॉ. प्रविण प्रकाश आडिवरेकर, ए. पी. शाह इन्स्टिट्यूट ऑफ टेक्नॉलॉजी, यांनी केलेल्या मराठी भाषेतील पुनरावलोकनासाठी त्यांचे विशेष आभार व्यक्त करतो.

IIT बॉम्बे मधील प्रा. गणेश रामकृष्णन यांच्या नेतृत्वाखालील UDAAN प्रोजेक्ट टीम (<https://www.udaanproject.org/>) यांना, पोस्ट-एडिटिंग प्लॅटफॉर्मसह त्यांचे नाविन्यपूर्ण एंड-टू-एंड मशीन भाषांतर फ्रेमवर्क प्रदान केल्याबद्दल आम्ही आभारी आहोत. त्यांच्या अमूल्य पाठिंब्यामुळे मूळ पुस्तक (डिप्लोमा ३ रे सत्र, संगणक अभियांत्रिकी।) मराठी मध्ये अनुवादित करण्यात खूप मदत झाली. या भाषांतर प्रकल्पाच्या यशात महत्वाची भूमिका बजावणाऱ्या त्यांच्या योगदानाबद्दल आणि कौशल्याबद्दल आम्ही खरोखरच कृतज्ञ आहोत.

शेवटी आम्ही या पुस्तकाचे प्रकाशक, खन्ना बुक पब्लिशिंग कंपनी प्रायव्हेट लिमिटेड, नवी दिल्ली, यांचे मनापासून आभार व्यक्त करू इच्छितो. ज्याची संपूर्ण टीम प्रकाशनाच्या सर्व पैलूंवर सहकार्य करण्यास सदैव तत्पर होती, ज्यामुळे या पुस्तक प्रकाशनाचा एक अद्भुत अनुभव आम्हाला घेता आला.

डॉ. सोनल यादव

प्रस्तावना

'संगणक प्रणाली संघटन' हे पुस्तक संगणक वास्तुकलेच्या मूलभूत आणि प्रगत अभ्यासक्रमांच्या आपल्या शिकवणीच्या समृद्ध अनुभवाचा परिणाम आहे.

हे पुस्तक लिहिण्याची सुरुवात अभियांत्रिकी विद्यार्थ्यांसमोर संगणक प्रणाली संघटनेची मूलभूत तत्त्वे, संगणकाच्या हार्डवेअरशी संवाद साधण्यासाठी असेंब्ली प्रोग्रामिंगचा वापर उघड करणे ही आहे. व्यापक व्याप्तीचा उद्देश लक्षात घेऊन तसेच आवश्यक पूरक माहिती प्रदान करण्यासाठी, आम्ही ए. आय. सी. टी. ई. ने शिफारस केलेले विषय संपूर्ण पुस्तकात अतिशय पद्धतशीर आणि सुव्यवस्थित पद्धतीने समाविष्ट केले आहेत. या विषयाच्या मूलभूत संकल्पना शक्य तितक्या सोप्या पद्धतीने स्पष्ट करण्याचे प्रयत्न केले गेले आहेत.

पुस्तक तयार करण्याच्या प्रोसीजरदरम्यान, आम्ही विविध मानक पाठ्यपुस्तकांचा विचार केला आहे आणि त्यानुसार आम्ही विभाग आणि बहुपर्यायी, लहान आणि लांब उत्तरे, संख्यात्मक समस्या आणि पूरक साहित्य यासारखे विविध प्रश्न विकसित केले आहेत. विविध विभाग तयार करताना उदाहरणे, पूरक साहित्य आणि मूलभूत तत्त्वांच्या त्वरित पुनरावलोकनासाठी प्रमुख मुद्द्यांच्या सर्वसमावेशक सारांशावर देखील भर देण्यात आला आहे. या पुस्तकात सर्व प्रकारच्या मध्यम आणि प्रगत पातळीवरील समस्यांचा समावेश आहे आणि त्या अतिशय लॉजिक आणि पद्धतशीर पद्धतीने सादर केल्या आहेत. त्या समस्यांच्या श्रेणीकरणाची चाचणी अनेक वर्षांच्या अध्यापनानंतर विविध प्रकारच्या विद्यार्थ्यांना दिली गेली आहे.

आवश्यकतेनुसार चित्रे आणि उदाहरणांव्यतिरिक्त, संबंधित विषयांच्या योग्य समजुतीसाठी आम्ही प्रत्येक युनिटमध्ये असंख्य सोडवलेल्या समस्यांसह पुस्तक समृद्ध केले आहे. "संगणक प्रणाली संघटना" या सामान्य शीर्षकाखाली अभियांत्रिकीमधील संगणकाचे विविध पैलू आणि संघटना यांचा समावेश असलेल्या पाच अध्यायांचा सेट आहे.

पहिल्या अध्यायात संगणकाची रचना समाविष्ट आहे, ज्यानंतर मायक्रोप्रोग्राम कंट्रोल, मायक्रोप्रोसेसर आर्किटेक्चर, असेंब्ली लॅंग्वेज प्रोग्रामिंग, मेमोरी आणि डिजिटल इंटरफेसिंगवरील अध्याय आहेत. हे लक्षात घेणे महत्वाचे आहे की सर्व अध्यायांमध्ये संबंधित प्रयोगशाळा व्यावहारिक आहेत. याव्यतिरिक्त, "अधिक जाणून घ्या" या शीर्षकाखाली वापरकर्त्यासाठी काही आवश्यक माहिती व्यतिरिक्त आम्ही उल्लेखनीय भारतीय शोधक तसेच समृद्ध भारतीय वेद ज्ञान आणि मूलभूत तत्त्वे सादर करतो जेणेकरून वाचकांना आधुनिक जीवनशैलीमध्ये आपली मौल्यवान तत्त्वे पाळण्यास प्रेरित करता येईल.

सध्याच्या पुस्तकाचा विचार करता, 'संगणक प्रणाली संघटन' चा उद्देश समाविष्ट केलेल्या विषयांवर कॉम्प्युटर आर्किटेक्चरमध्ये सखोल आधार प्रदान करणे हा आहे. संगणक प्रणाली संघटना पुस्तकाचा हा भाग अभियांत्रिकी विद्यार्थ्यांना 21 व्या शतकातील आणि पुढील अभियांत्रिकी आव्हानांचा सामना करण्यासाठी आणि संबंधित उद्भवलेल्या प्रश्नांची उत्तरे देण्यासाठी संगणक वास्तुकलेचे ज्ञान लागू करण्यासाठी तयार करेल. विषयाच्या बाबी रचनात्मक पद्धतीने सादर केल्या जातात जेणेकरून अभियांत्रिकी पदवी विद्यार्थ्यांना विविध फील्डमध्ये किंवा राष्ट्रीय प्रयोगशाळांमध्ये तंत्रज्ञानाच्या आघाडीवर काम करण्यासाठी तयार करते.

आम्हाला प्रामाणिकपणे आशा आहे की हे पुस्तक विद्यार्थ्यांना संगणक प्रणाली संघटनेच्या मूलभूत तत्त्वांमागील कल्पना शिकण्यासाठी आणि त्यावर चर्चा करण्यासाठी प्रेरित करेल आणि विषयाचा भक्कम पाया

विकसित करण्यात नक्कीच योगदान देईल. पुस्तकाच्या भविष्यातील आवृत्त्यांच्या सुधारणेला हातभार लावणाऱ्या सर्व फायदेशीर टिप्पण्या आणि इंस्ट्रक्शनसाठी आम्ही आभारी राहू. हे पुस्तक शिक्षक आणि विद्यार्थ्यांच्या हातात देताना आम्हाला खूप आनंद होतो. या पुस्तकात समाविष्ट असलेल्या विविध पैलूंवर काम करणे खरोखरच खूप आनंददायक होते.

डॉ. सोनल यादव

आऊटकम आधारित शिक्षण

परिणामांवर आधारित शिक्षणाच्या अंमलबजावणीसाठी पहिली आवश्यकता म्हणजे परिणामांवर आधारित अभ्यासक्रम विकसित करणे आणि शिक्षण व्यवस्थेत परिणामांवर आधारित व्हॅल्यूकनाचा समावेश करणे. परिणामांवर आधारित व्हॅल्यूकनांमधून जाऊन व्हॅल्यूकनकर्ते हे व्हॅल्यूकन करू शकतील की विद्यार्थ्यांनी निर्धारित मानक, विशिष्ट आणि मोजता येण्याजोगे परिणाम साध्य केले आहेत की नाही. परिणामांवर आधारित शिक्षणाचा योग्य प्रकारे समावेश केल्यास कोणत्याही स्तरावर हार न मानता सर्व विद्यार्थ्यांसाठी किमान मानक साध्य करण्याची फिक्स्ड बांधिलकी असेल. परिणामांवर आधारित शिक्षणाच्या मदतीने चालवल्या जाणाऱ्या प्रोग्रॅमच्या शेवटी, विद्यार्थी खालील परिणामांवर पोहोचू शकेल:

PO1. मूलभूत आणि शिस्त विशिष्ट ज्ञान: अभियांत्रिकी समस्यांचे निराकरण करण्यासाठी मूलभूत गणित, विज्ञान आणि अभियांत्रिकी मूलतत्त्वे आणि अभियांत्रिकी विशेषतेचे ज्ञान लागू करा.

PO2. समस्या विश्लेषण: संहिताबद्ध मानक पद्धतींचा वापर करून चांगल्या प्रकारे परिभाषित अभियांत्रिकी समस्या ओळखा आणि त्यांचे विश्लेषण करा.

PO3. उपायांची रचना/विकास: सुस्पष्ट तांत्रिक समस्यांसाठी उपायांची रचना करणे आणि विशिष्ट गरजा पूर्ण करण्यासाठी प्रणाली घटक किंवा प्रक्रियांच्या रचनेत मदत करणे.

PO4. अभियांत्रिकी साधने, प्रयोग आणि चाचणी: प्रमाणित चाचण्या आणि मोजमाप करण्यासाठी आधुनिक अभियांत्रिकी साधने आणि योग्य तंत्र लागू करा.

PO5. समाज, शाश्वतता आणि पर्यावरणासाठी अभियांत्रिकी पद्धती: समाज, शाश्वतता, पर्यावरण आणि नैतिक पद्धतींच्या संदर्भात योग्य तंत्रज्ञान लागू करा.

PO6. प्रकल्प व्यवस्थापन: प्रकल्प व्यवस्थापित करण्यासाठी आणि चांगल्या प्रकारे परिभाषित अभियांत्रिकी उपक्रमांबद्दल प्रभावीपणे संवाद साधण्यासाठी संघ सदस्य किंवा नेता म्हणून अभियांत्रिकी व्यवस्थापन तत्त्वांचा वैयक्तिकरित्या वापर करा.

PO7. आयुष्यभर शिकणे: वैयक्तिक गरजांचे विश्लेषण करण्याची आणि तांत्रिक बदलांच्या संदर्भात अद्ययावत करण्यात गुंतण्याची क्षमता

अभ्यासक्रमाचे परिणाम

अभ्यासक्रम पूर्ण केल्यानंतर, विद्यार्थ्यांना हे करता येईल:

CO-1: संगणक प्रणालीच्या कामकाजाची चांगली समज असणे.

CO-2: संगणकाच्या विविध उपघटकांच्या कार्याची चांगली समज असणे.

CO-3: विद्यार्थी विशिष्ट हेतूसाठी संगणकीय आवश्यकता समजून घेण्यास सक्षम असतील.

CO-4: संगणकीय मशीन कामगिरीतील अडथळ्यांचे विश्लेषण करणे.

CO-5: दिलेल्या वापरासाठी योग्य संगणकीय साधन निवडीने.

प्रोग्रॅमच्या परिणामांसह अभ्यासक्रमाच्या परिणामांचे मापिंग खाली दिलेल्या माट्रिक्सनुसार केले जाईल:

अभ्यासक्रम चे परिणाम	प्रोग्रॅमच्या परिणामा सह अपेक्षित मापिंग 1-कमकुवत मापिंग, 2- मध्यम मापिंग, 3- मजबूत मापिंग						
	PO-1	PO-2	PO-3	PO-4	PO-5	PO-6	PO-7
CO-01	3	3	3	3	1	2	3
CO-02	3	3	3	3	1	2	3
CO-03	3	3	3	3	1	2	3
CO-04	3	3	2	3	1	2	3
CO-05	3	3	2	3	1	1	3

शिक्षकांसाठी मार्गदर्शक तत्त्वे

परिणामांवर आधारित शिक्षणाची (ओ. बी. ई.) अंमलबजावणी करण्यासाठी विद्यार्थ्यांच्या ज्ञानाची पातळी आणि कौशल्य वाढवणे आवश्यक आहे. ओ. बी. ई. च्या योग्य अंमलबजावणीसाठी शिक्षकांनी मोठी जबाबदारी घेतली पाहिजे. ओ. बी. ई. प्रणालीतील शिक्षकांच्या काही जबाबदाऱ्या (इतकेच मर्यादित नाहीत) खालीलप्रमाणे असू शकतात:

- वाजवी मर्यादित, सर्व विद्यार्थ्यांच्या सर्वोत्तम फायद्यासाठी वेळेचा वापर केला पाहिजे.
- त्यांनी भेदभाव करण्याच्या इतर कोणत्याही संभाव्य अपात्रतेचा विचार न करता केवळ काही फिक्स्ड निकषांवर विद्यार्थ्यांचे मूल्यांकन केले पाहिजे.
- संस्थेतून बाहेर पडण्यापूर्वी त्यांनी विद्यार्थ्यांची शिकण्याची क्षमता एका विशिष्ट स्तरावर वाढविण्याचा प्रयत्न केला पाहिजे.
- त्यांनी हे सुफिक्स्ड करण्याचा प्रयत्न केला पाहिजे की सर्व विद्यार्थी त्यांचे शिक्षण पूर्ण केल्यानंतर दर्जेदार ज्ञान आणि क्षमता यांनी सुसज्ज आहेत. त्यांनी नेहमीच विद्यार्थ्यांना त्यांची अंतिम कामगिरी क्षमता विकसित करण्यासाठी प्रोत्साहित केले पाहिजे.
- त्यांनी नवीन दृष्टीकोन एकत्रित करण्यासाठी समूह कार्य आणि सांघिक कार्य सुलभ आणि प्रोत्साहित केले पाहिजे.
- त्यांनी मूल्यांकनच्या प्रत्येक भागात ब्लूमच्या वर्गीकरणाचे पालन केले पाहिजे.

ब्लूम वर्गीकरण

स्तर	शिक्षकांनी तपासावे	विद्यार्थी सक्षम असावा	मूल्यांकनाची संभाव्य पद्धत
निर्माण करणे	विद्यार्थी तयार करण्याची क्षमता	डिझाइन करा किंवा तयार करा	सूक्ष्म प्रकल्प
मूल्यमापन	विद्यार्थ्यांचे औचित्य सिद्ध करण्याची क्षमता	वाद घालणे किंवा बचाव करणे	असाइनमेंट
विश्लेषण करणे	विद्यार्थ्यांमध्ये फरक करण्याची क्षमता	फरक किंवा भेद करा	प्रकल्प/प्रयोगशाळा पद्धती
अर्ज करणे	विद्यार्थ्यांची माहिती वापरण्याची क्षमता	चालवा किंवा प्रात्यक्षिक करा	तांत्रिक सादरीकरण/ प्रात्यक्षिक
समजून घेणे	विद्यार्थ्यांची कल्पना स्पष्ट करण्याची क्षमता	स्पष्ट करा किंवा वर्गीकृत करा	सादरीकरण / परिसंवाद
आठवणे	विद्यार्थ्यांची आठवण करण्याची क्षमता (किंवा लक्षात ठेवणे)	व्याख्या करा किंवा आठवा	प्रश्नमंजुषा

विद्यार्थ्यांसाठी मार्गदर्शक तत्त्वे

ओ. बी. ई. च्या अंमलबजावणीसाठी विद्यार्थ्यांनी समान जबाबदारी घेतली पाहिजे. ओ. बी. ई. प्रणालीतील विद्यार्थ्यांच्या काही जबाबदाऱ्या (इतकेच मर्यादित नाहीत) खालीलप्रमाणे आहेत:

- प्रत्येक अभ्यासक्रमात एक युनिट सुरु होण्यापूर्वी विद्यार्थ्यांना प्रत्येक UO ची चांगली माहिती असली पाहिजे.
- अभ्यासक्रम सुरु होण्यापूर्वी विद्यार्थ्यांना प्रत्येक CO ची चांगली माहिती असली पाहिजे.
- विद्यार्थ्यांना प्रोग्राम सुरु होण्यापूर्वी प्रत्येक PO ची चांगली माहिती असली पाहिजे.
- विद्यार्थ्यांनी योग्य चिंतन आणि कृतीसह गंभीरपणे आणि तर्कशुद्धपणे विचार केला पाहिजे.
- विद्यार्थ्यांचे शिक्षण व्यावहारिक आणि वास्तविक जीवनातील परिणामांशी जोडले गेले पाहिजे आणि समाकलित केले गेले पाहिजे.
- विद्यार्थ्यांना ओ. बी. ई. च्या प्रत्येक स्तरावर त्यांच्या क्षमतेची चांगली जाणीव असली पाहिजे.

संक्षिप्तरूपे आणि चिन्हे

संक्षेपांची यादी

साध्या परिलिंगवेज

संक्षिप्तरूपे	पूर्ण रूप	संक्षिप्तरूपे	पूर्ण रूप
AC	अकुमुलेटर रजिस्टर	DS	डेटा सेगमेंट
ADC	अनालॉग टू डिजिटल कन्व्हर्जण	DVD	डिजिटल व्हर्सेटाइल डिस्क
ALSU	अरीथमेटिक लॉजिक शिफ्ट युनिट	DX	डेटा रजिस्टर
ALU	अरीथमेटिक लॉजिक युनिट	ES	एक्सट्रा सेगमेंट
ASHL	अरीथमेटिक शिफ्ट लेफ्ट	EU	एक्सिक्युशन युनिट
ASHR	अरीथमेटिक शिफ्ट राइट	FIFO	फर्स्ट इन फर्स्ट आऊट
AX	अकुमूलॅटोर रजिस्टर	HDD	हार्ड डिस्क ड्राईव्ह
BHT	ब्रांच हिस्टरी टेबल	I/O	इनपुट/आउटपुट
BIOS	बेसिक इनपुट/आउटपुट सिस्टिम	IP	इंस्ट्रक्शन पॉईंटर
BIU	बस इंटरफेस युनिट	IR	इंस्ट्रक्शन रजिस्टर
BP	बेस पॉइंटर	ISA	इंस्ट्रक्शन सेट आर्कीटेक्चर
BSR	बिट सेट/रीसेट	ISZ	इन्क्रेमेन्ट आणि स्किप इफ झिरो
BX	बेस रजिस्टर	LSB	लीस्ट सिग्निफिकन्ट बिट
CAR	कंट्रोल ऍड्रेस रजिस्टर	MAR	मेमोरी ऍड्रेस रजिस्टर
CD	कॉम्पॅक्ट डिस्क	MDR	मेमोरी डेटा रजिस्टर
Cil	सर्क्युलर शिफ्ट लेफ्ट	MICR	माग्नेटिक इंक कॅरॅक्टर रेकग्नाशन
Cir	सर्क्युलर शिफ्ट राइट	MMU	मेमोरी मानॅजमेण्ट युनिट
CISC	कॉम्प्लेक्स इंस्ट्रक्शन सेट कॉम्पुटिंग	MSB	मोस्ट सिग्निफिकन्ट बिट
CPU	सेंट्रल प्रोसेसिंग युनिट	NASM	नेटवाईड असेम्बलर
CS	कोड सेगमेन्ट	NOP	नो ओपेरेशन
CU	कंट्रोल युनिट	OCR	ऑप्टिकल कॅरॅक्टर रेकग्निशन
CX	काउन्ट रजिस्टर	PC	प्रोग्रॅम काउंटर
DAC	डिजिटल टू अनालॉग कन्व्हर्जण	PROM	प्रोग्रामबल रीड ओन्ली मेमोरी
DB	डिफाइन बाइट	RAM	रॅंडम ऍक्सेस मेमोरी
DI	डेस्टीनेशन इंडेक्स	RAW	रीड आफ्टर राइट
RISC	रिड्युस इंस्ट्रक्शन सेट कॉम्पुटिंग	SP	स्टॅक पॉईंटर

ROM	रीड ओन्ली मेमोरी	SS	स्टॉक सेगमेंट
RTL	रजिस्टर ट्रान्सफर लॉगवेज	SSD	सॉलिड स्टेट डिवाइस
SC	सिक्वेन्स काउंटर	USB	युनिवर्सल सिरीयल बस
SD	सेक्युअर डिजिटल	WAR	राइट आफ्टर रीड
Shl	लॉजिकल शिफ्ट लेफ्ट	WAW	राइट आफ्टर राइट
Shr	लॉजिकल शिफ्ट राइट	WMFC	वेट फॉर मेमोरी फंक्शन कंप्लिट
SI	सोर्स इंडेक्स	ZF	झिरो फ्लॉग
SIMD	सिंगल इंस्ट्रक्शन मल्टीपल डेटा		

एककांची यादी

संक्षिप्तरूपे	साध्या परिल्लेखेज		पूर्ण रूप
	पूर्ण रूप	संक्षिप्तरूपे	
GB	गिगा बाइट	ns	नॅनो सेकंड
KB	किलो बाइट	TB	टेरा बाइट

चिन्हांची यादी

चीन्ह	वर्णन	चीन्ह	वर्णन
+	बेरीज	x	गुणाकार
^	अँड ऑपरेशन	A'	नॉट A
/	भागाकार	V	ऑर ऑपरेशन
X	डोन्ट केयर	-	वजाबाकी
*	गुणाकार	⊕	एक्सऑर ऑपरेशन

आकृत्यांची यादी

आकृती क्रमांक	आकृती चे शीर्षक	पृष्ठ क्रमांक
युनिट 1. संगणकाची रचना		
आकृती. 1.1: संगणक कार्यात्मक एकक		4
आकृती. 1.2: इनपुट उपकरणे		5
आकृती. 1.3: मेमोरी पदानुक्रम: रजिस्टर, कॅशे आणि मुख्य मेमोरी ही अस्थिर मेमोरी आणि सॉलिड स्टेट ड्राइव्ह आहेत, यांत्रिक हार्ड ड्राइव्ह ही अस्थिर नसलेली मेमोरी आहेत.		6
आकृती. 1.4: कॅशे मेमोरीचे स्थाननिर्धारण		7
आकृती. 1.5: दुय्यम स्टोरेज		8
आकृती. 1.6: एरिथमेटिक लॉजिक एकक (ALU)		9
आकृती. 1.7: कंट्रोल युनिट इंस्ट्रक्शन डीकोड करते आणि मास्टर क्लॉकद्वारे कंट्रोल सिग्नल तयार करते इव्हेंट सिंक्रोनाइझ करा (IR: इंस्ट्रक्शन रजिस्टर)		10
आकृती. 1.8: इंटरकनेक्शन नेटवर्क		11
आकृती. 1.9: आउटपुट युनिट		12
आकृती. 1.10: वॉन-न्युमन आर्किटेक्चर		12
आकृती. 1.11: सिस्टम बसची रचना डेटा बस, अ‍ॅड्रेस बस आणि कंट्रोल बस यांनी बनलेली आहे		14
आकृती. 1.12: प्रोसेसर रजिस्टर्स आणि मुख्य मेमोरीची रचना		14
आकृती. 1.13: 8-बिट बायनरी नंबरमध्ये MSB आणि LSB बिट्स		16
आकृती. 1.14: साइन आणि माग्नीटुड, 1's कॉम्प्लिमेंट आणि 2s कॉम्प्लिमेंट संख्या प्रणालींमध्ये + 64 आणि -64 साइन केलेल्या संख्यांचे प्रतिनिधित्व		18
आकृती. 1.15: दोन सकारात्मक संख्या +64 आणि +84 जोडताना ओव्हरफ्लो डिटेक्शन		19
आकृती. 1.16: दोन ऋण संख्या -64 आणि -84 जोडल्यावर ओव्हरफ्लो डिटेक्शन		21
आकृती. 1.17: रजिस्टरमधील व्हॅल्यूचे भिन्न प्रतिनिधित्व		25
आकृती. 1.18: रजिस्टरमधून बसद्वारे माहितीचे ट्रान्सफर		27
आकृती. 1.19: अरीथमेटिक आणि लॉजिक शिफ्ट युनिट मायक्रोऑपरेशन्स योजनाबद्ध आकृती आणि कार्य सारणी		29
आकृती. 1.20: लॉजिक मायक्रो ऑपरेशन आणि फंक्शन टेबलची योजनाबद्ध आकृती		32
आकृती. 1.21: लॉजिकल शिफ्ट (अ) डावे आणि (ब) उजवे सूक्ष्म ऑपरेशन		35
आकृती. 1.22: एरिथमेटिक शिफ्ट (अ) डावीकडे आणि (ब) उजवीकडे सूक्ष्म कार्ये		35
आकृती. 1.23: सर्क्युलर शिफ्ट (अ) डावे आणि (ब) उजवे मायक्रो ऑपरेशन्स		36

आकृती. 2.1: कंट्रोल मेमोरी अँड्रेस सिलेक्शन	52
आकृती. 2.2: कंट्रोल युनिट ब्लॉक डायग्रॅम	54
आकृती. 2.3: हार्डवायर्ड कंट्रोल युनिट	55
आकृती. 2.4: मायक्रोप्रोग्राम कंट्रोल ऑर्गनायझेशन	56
आकृती. 2.5: साइन आणि माग्रीटूड संख्या साठी बेरीज आणि वजाबाकी ऑपरेशन ची हार्डवेअर अंमलबजावणी	58
आकृती. 2.6: साइन आणि माग्रीटूड बेरीज आणि वजाबाकी ऑपरेशन चा फ्लोचार्ट	59
आकृती. 2.7: 2 कॉम्प्लिमेंट संख्यांसाठी बेरीज आणि वजाबाकी च्या हार्डवेअर ची अंमलबजावणी	60
आकृती. 2.8: 2 कॉम्प्लिमेंट संख्यांसाठी बेरीज आणि वजाबाकी चा फ्लोचार्ट	60
आकृती. 2.9: साइन आणि माग्रीटूड संख्या गुणाकार	61
आकृती. 2.10: साइन आणि माग्रीटूड गुणाकार फ्लोचार्ट	62
आकृती. 2.11: बूथ गुणन यंत्रसामग्री कार्यान्वयन	63
आकृती. 2.12: बूथचा गुणाकार फ्लोचार्ट	64
आकृती. 2.13: रिस्टोरिंग भागाकार फ्लोचार्ट	65
आकृती. 2.14: नॉन रिस्टोरिंग भागाकार	67
आकृती. 2.15: फ्लोटिंग पॉइंट अरिथमेटिक ऑपरेशन्ससाठी रजिस्टर	71
आकृती. 2.16: फ्लोटिंग पॉइंट बायनरी नंबरची बेरीज किंवा वजाबाकी	72
आकृती. 2.17: फ्लोटिंग पॉइंट बायनरी नंबर्सचा गुणाकार	73
आकृती. 2.18: माग्रीटूड माग्रीटूड भागाकार	74
आकृती. 2.19: फ्लोटिंग पॉइंट बायनरी नंबर्स चा भागाकार	74
आकृती. 2.20: फ्लोटिंग पॉइंट अँडर ची पाईपलाईन अंमलबजावणी	75
आकृती. 2.21: इंस्ट्रक्शन पाईपलाईन अंमलबजावणी	76
आकृती. 2.22: रेसोर्स किंवा संरचनात्मक हझार्ड	77
आकृती. 2.23: संरचनात्मक/रेसोर्स हझार्ड सोलुशन	77
आकृती. 2.24: इंस्ट्रक्शन पाईपलाईन अंमलबजावणीमध्ये रीड आफ्टर राईट हझार्ड	78
आकृती. 2.25: RAW इंस्ट्रक्शन हझार्ड चे उपाय	78
आकृती. 2.26: कंट्रोल हझार्ड परिस्थिती	81
आकृती. 2.27: कंट्रोल हझार्ड सोलुशन	82
आकृती. 2.28: 1-बिट डायनॅमिक ब्रांच प्रेडिक्शनची स्थिती आकृती	83
आकृती. 2.29: हार्डवेअर प्रेडिक्शन आणि वास्तविक प्रेडिक्शन	84

युनिट 3. मायक्रोप्रोसेसर आर्किटेक्चर	
आकृती. 3.1: ऑपरेंड ऍड्रेस रजिस्टरमध्ये साठवला जातो आणि ऍड्रेस मेमोरीमध्ये साठवला जातो तेव्हा अप्रत्यक्ष ऍड्रेसिंग मोडची रजिस्टर करा.	109
आकृती. 3.2: इंडेक्स ऍड्रेसिंग मोड मधील मुख्य मेमोरी आणि रजिस्टर कंटेन्ट.	110
आकृती. 3.3: बेस-इंडेक्स आणि बेस-इंडेक्स ऑफसेट ऍड्रेसिंग मोडमध्ये मुख्य मेमोरी आणि रजिस्टर सामग्री	111
आकृती. 3.4: रजिस्टर ऍड्रेसमध्ये ऑटोइन्क्रीमेंट आणि ऑटोडीक्रीमेंट ऍड्रेसिंग मोडसह सुधारणा	113
आकृती. 3.5: 8086 मायक्रोप्रोसेसर आर्किटेक्चर	115
युनिट 4. असेम्ब्ली लॉगवेज प्रोग्रामिंग	
आकृती. 4.1: मशीनी भाषेत असेम्ब्ली लॉगवेज ADD डारेक्टिव प्रतिनिधित्व	132
युनिट 5. मेमोरी आणि डिजिटल इंटरफेसिंग	
आकृती. 5.1: मेमोरी आणि आय/ओ इंटरफेस	168
आकृती. 5.2: एक्सटर्नल उपकरणे	168
आकृती. 5.3: आय/ओ मॉड्यूलची ब्लॉक आकृती	169
आकृती. 5.4: सेमीकंडक्टर मेमोरी वर्गीकरण	170
आकृती. 5.5: 7-ऍड्रेस लाईन्स आणि 8-द्विदिशात्मक डेटा लाईन्स असलेली एक विशिष्ट रॅम चिप	171
आकृती. 5.6: सामान्यतः 4096 मुख्य मेमोरी ब्लॉक्सचे 128 कॅशे ब्लॉक्सचे थेट कॅशे ब्लॉक मार्पिंग	172
आकृती. 5.7: असोसिएटिव्ह मार्पिंग	175
आकृती. 5.8: सेट-असोसिएटिव्ह मार्पिंग	176
आकृती. 5.9: मेमोरी व्यवस्थापन युनिटचा वापर करून भौतिक ऍड्रेसच्या मार्पिंगसाठी लॉजिक ऍड्रेस	179
आकृती. 5.10: 9-ऍड्रेस लाईन्स आणि 8-युनिडायरेक्शनल डेटा लाईन्ससह एक टिपिकल रॉम चिप	180
आकृती. 5.11: फ्लोचार्ट प्रोग्राम केलेले I/O साठी	185
आकृती. 5.12: इंटरप्ट ड्रिव्हन I/O फ्लो	185

तक्त्याची यादी

तक्ता क्रमांक	तक्ताचे शीर्षक	पृष्ठ क्रमांक
तक्ता 1.1:	पॉजीटीव्ह आणि नेगेटिव्ह बायनरी नंबर, साइन आणि माग्नीटूड, 1s कॉम्पलिमेन्ट आणि 2s कॉम्पलिमेन्ट संख्येचे प्रतिनिधित्व	17
तक्ता 1.2:	सम आणि विषम पॅरिटी कोड प्रतिनिधित्व	24
तक्ता 1.3:	रजिस्टर ट्रान्सफरचे सिम्बॉलिक रेप्रेसेंटेशन	26
तक्ता 1.4:	अरीथमेटिक सूक्ष्म क्रिया	31
तक्ता 2.1:	साइन आणि माग्नीटूड संख्या गुणाकार उदाहरण (-6x3)	62
तक्ता 2.2:	बूथ गुणनाचे उदाहरण (-5x-4)	64
तक्ता 2.3:	रिस्टोरिंग भागाकार अल्गोरिदमचे उदाहरण	66
तक्ता 2.4:	नॉन रिस्टोरिंग भागाकार करण्याचे उदाहरण	68
तक्ता 4.1	प्रोसिजर आणि माक्रो यांच्यातील तुलना दर्शवितो	144
तक्ता 4.2:	एरिथमेटिक क्रियांची यादी	147
तक्ता 4.3:	लॉजिकल ऑपरेशन ची यादी	149
तक्ता 5.1:	SRAM आणि DRAM मधील फरक	171
तक्ता 5.2:	पोर्ट निवडीसाठी ऍड्रेसच्या रेषांची बिट व्हॅल्यू	182
तक्ता 5.3:	बीएसआर पद्धतीसाठी कंट्रोल रजिस्टर स्वरूप	183
तक्ता 5.4:	पोर्ट C ची पिन निवड	183

अनुक्रमिका

प्रास्ताविक	iv
ऋ णनदेश	v
प्रस्तावना	vi
आऊटकम आधारित शिक्षण	viii
अभ्यासक्रमाचे परिणाम	ix
शिक्षकांसाठी मार्गदर्शक तत्त्वे	x
विद्यार्थ्यांसाठी मार्गदर्शक तत्त्वे	xi
संक्षिप्तरूपे आणि चिन्हे	xii
आकृत्यांची यादी	xiv
तक्त्याची यादी	xvii
युनिट 1. संगणकाची रचना	1-48
युनिटची वैशिष्ट्ये	1
तर्क	1
पूर्व-आवश्यकता	2
युनिट निष्पत्ती	2
1.1 डिजिटल संगणक	2
1.2 संगणक कार्यात्मक युनिट्स	3
1.2.1 इनपुट युनिट्स	4
1.2.2 मेमोरी	5
1.2.3 अरीथमेटिक लॉजिक युनिट	8
1.2.4 कंट्रोल युनिट	9
1.2.5 इंटरकनेक्शन	10
1.2.6 आउटपुट युनिट	11
1.3 वॉन-न्युमन आर्किटेक्चर	12
1.4 बस संरचना	13
1.5 मूलभूत ऑपरेशनल संकल्पना	14
1.6 डेटा प्रतिनिधित्व	16
1.6.1 फिक्स्ड पॉइंट पूर्णांक प्रतिनिधित्व	16
1.6.2 फ्लोटिंग पॉइंट क्रमांकाचे प्रतिनिधित्व	21

1.7 एरर डिटेक्शन कोड	23
1.8 रजिस्टर ट्रान्सफर आणि सूक्ष्म ऑपरेशन्स	25
1.8.1 रजिस्टर ट्रान्सफर	25
1.8.2 बस आणि मेमोरी ट्रान्सफर	27
1.8.3 अरीथमेटिक आणि शिफ्ट लॉजिक युनिट	30
युनिट सारांश	37
स्वाध्याय:	38
प्रात्याक्षिक	45
अधिक जाणून घ्या	46
संदर्भ आणि सुचविलेले वाचन	47
युनिट 2. मायक्रो प्रोग्रॅम कंट्रोल	49-97
युनिटची वैशिष्ट्ये	49
तर्क	49
पूर्व-आवश्यकता	50
युनिट निष्पत्ती	50
2.1 कंट्रोल मेमोरी	51
2.2 ऍड्रेस सिकवेन्ससिंग	51
2.3 कंट्रोल युनिट डीजाइन	53
2.3.1 हार्डवायर्ड कंट्रोल युनिट	54
2.3.2 मायक्रोप्रोग्राम कंट्रोल युनिट	55
2.4 संगणक अरीथमेटिक	57
2.4.1 साइन आणि माग्रीटूड संख्यांसाठी बेरीज आणि वजाबाकी	57
2.4.2 2s कॉम्प्लिमेंट संख्यांची बेरीज व वजाबाकी	59
2.4.3 पूर्णांक गुणाकार	60
2.4.4 पूर्णांक भागाकार	65
2.5 अपूर्णांक संख्या प्रतिनिधित्व	68
2.5.1 फिक्स्ड पॉइंट प्रतिनिधित्व	68
2.5.2 फ्लोटिंग पॉइंट प्रतिनिधित्व	69
2.5.3 फ्लोटिंग पॉइंट एरिथमेटिक ऑपरेशन्स	70
2.6 अरीथमेटिक पाइपलाइन	74
2.7 इंस्ट्रक्शन पाइपलाइन	75

2.7.1 रेसोर्स किंवा संरचनात्मक हार्ड	76
2.7.2 डेटा हार्ड	78
2.7.3 कंट्रोल हार्ड	81
2.8 रिस्क पाईपलाईन	84
2.9 वेक्टर प्रोसेसिंग	85
2.10 अरे प्रोसेसर	85
युनिट सारांश	87
स्वाध्याय	89
प्रात्याक्षिक	94
अधिक जाणून घ्या	95
संदर्भ आणि सुचविलेले वाचन	96
युनिट 3. मायक्रोप्रोसेसर आर्किटेक्चर	98-129
युनिटची वैशिष्ट्ये	98
तर्क	98
पूर्व-आवश्यकता	99
युनिट निष्पत्ती	99
3.1 परिचय	99
3.2 इंस्ट्रक्शन सेट आर्किटेक्चर	103
3.2.1 CISC ची वैशिष्ट्ये	103
3.2.2 RISC ची वैशिष्ट्ये	104
3.3 अभ्यागत प्रोग्रॅमची रचना तत्त्वे	105
3.3.1 इंस्ट्रक्शन फॉरमाट	105
3.3.2 अड्रेसिंग मोड	106
3.4 8086 मायक्रोप्रोसेसर आर्किटेक्चर	113
3.4.1. 8086 मायक्रोप्रोसेसर कार्यात्मक एकक	113
3.4.2 8086 मधील इंस्ट्रक्शन चे प्रकार	118
युनिट सारांश	120
स्वाध्याय	121
प्रात्याक्षिक	127
अधिक जाणून घ्या	128
संदर्भ आणि सुचविलेले वाचन	129

युनिट 4. असेम्बली लाँगवेज प्रोग्रामिंग	130-165
युनिटची वैशिष्ट्ये	130
तर्क	130
पूर्व-आवश्यकता	131
युनिट निष्पत्ती	131
4.1 परिचय	131
4.2 असेम्बली लाँगवेज प्रोग्रामिंग	133
4.2.1 NASM सह पहिला असेम्बली प्रोग्राम्स	135
4.3 असेम्बलर डारेक्टिव	140
4.4 प्रोसीजर आणि माक्रोस	142
4.4.1 प्रक्रिया	142
4.4.2 माक्रो	144
4.5 असेम्बली प्रोग्राम	145
4.5.1 सिम्पल प्रोग्राम	145
4.5.2 अरीथमाटिकस प्रोग्राम्स	147
4.5.3 लॉजिकल इंस्ट्रक्शन	149
4.5.4 ब्रँच इंस्ट्रक्शन	150
4.5.5 एरिथमेटिक एक्सप्रेसनचे व्हॅल्यूकन	152
4.5.6 स्ट्रिंग मानिपुलेशन	153
4.5.7 सॉर्टिंग	155
युनिट सारांश	160
स्वाध्याय	161
प्रात्याक्षिक	164
अधिक जाणून घ्या	164
संदर्भ आणि सुचविलेले वाचन	165
युनिट 5. मेमोरी आणि डिजिटल इंटरफेसिंग	166-196
युनिटची वैशिष्ट्ये	166
तर्क	166
पूर्व-आवश्यकता	167
युनिट निष्पत्ती	167
5.1 परिचय	167

5.2 मेमोरी प्रकार आणि वैशिष्ट्ये	169
5.2.1 मेमोरीचे प्रकार	169
5.2.2 रँडम ऍक्सेस मेमोरी (RAM)	170
5.2.3 कॅशे मेमोरी मारपिंग तंत्र	171
5.2.4 रीड ओन्ली मेमोरी (ROM)	179
5.3 सेकंडरी मेमोरी	180
5.4 प्रोग्रामेबल पेरिफेरल इंटरफेस	182
5.4.1 ऑपरेशनल मोड 8255	182
5.4.2 प्रोसेसरशी जोडणी	184
5.4.3 कीबोर्ड आणि डिस्प्ले डिव्हाइसेस	186
युनिट सारांश	187
स्वाध्याय	189
प्रात्याक्षिक	193
अधिक जाणून घ्या	194
संदर्भ आणि सुचविलेले वाचन	196
पुढील शिक्षणासाठी संदर्भ	197
सीओ आणि पीओ अटेंन्मेंट टेबल	198
इंडेक्स	199

1. संगणकाची रचना

युनिट ची वैशिष्ट्य (UNIT SPECIFIC)

या युनिटमध्ये खालील पैलूवर चर्चा केली आहे:

- संगणक कार्यात्मक युनिट्सची मूलभूत रचना;
- वॉन न्यूमन आणि हार्वर्ड आर्किटेक्चर;
- बस ची रचना आणि बसचे प्रकार
- प्रोसेसर ची मूलभूत ऑपरेशनल संकल्पना;
- संख्यात्मक आणि फ्लोटिंग पॉइंट संख्यांचे प्रतिनिधित्व आणि एरिथमेटिक ऑपरेशन्स;
- एरर डिटेक्शन च्या पद्धती;
- एरिथमेटिक, लॉजिक आणि शिफ्ट मायक्रो-ऑपरेशनसाठी ALU चे कार्य.

अधिक जिज्ञासा आणि सर्जनशीलता वाढवण्यासाठी आणि समस्या सोडवण्याची कौशल्ये वाढवण्याच्या उद्देशाने विषयांचे व्यावहारिक अनुप्रयोग सादर केलेले आहेत. मोठ्या संख्येच्या व्यतिरिक्त दोन श्रेणींमध्ये चिन्हांकित केलेले बहु-निवडीचे प्रश्न आणि लहान- कींवा दीर्घ-उत्तरांचे प्रश्न ब्लूमच्या वर्गीकरणाच्या खालच्या आणि उच्च स्तरांनुसार, युनिट सराव प्रदान करतात. संख्यात्मक समस्या, संदर्भांची सूची आणि सुचवलेले वाचन या स्वरूपात असाइनमेंट. हे लक्षात घेणे महत्वाचे आहे की स्वाभाविक असलेल्या विविध विषयांवरील अधिक माहितीसाठी स्कॅन केले जाऊ शकणारे अनेक क्यूआर कोड वेगवेगळ्या भागांमध्ये समाविष्ट केले गेले आहेत आणि आवश्यक आधार डेटा प्राप्त करण्यासाठी वापरले जाऊ शकतात.

सामग्रीवर आधारित संबंधित व्यावहारिक नंतर “*Know more*” विभाग आहे. हा विभाग काळजीपूर्वक तयार केला गेला आहे जेणेकरून समाविष्ट पुस्तका ची पूरक वाचकांसाठी मौल्यवान माहिती मिळेल. हा विभाग भारतीय नवसंशोधकांची संगणक प्रणाली संघटनेत योगदान आणि भारतीय वैदिक ज्ञानाचा प्रभाव समकालीन डिजिटल संगणक ची चर्चा करतो.

तर्क

या युनिटमध्ये संगणक प्रणालीचे कार्यात्मक घटक आणि उपघटकाकार्यात्मक घटकांबद्दल मूलभूत माहिती आहे. यातील वैशिष्ट्ये, कार्यप्रदर्शन आणि अनुप्रयोग घटकांची सखोल चर्चा केली गेली आहे. बस या घटकांना सिग्नल, पत्ता आणि डेटा एकमेकांसह तसेच स्मृतीसह प्रसारित आणि प्राप्त करण्यास सक्षम करतात. बेसिक बसेसची रचना आणि प्रकार देखील समाविष्ट केले गेले आहेत. पारंपारिक वॉन न्यूमनची रचना "सेटयित प्रोग्राम संकल्पना" वर आधारित आर्किटेक्चरची तुलना समकालीन हार्वर्ड आर्किटेक्चर डिझाइनशी केली आहे. साइन आणि अन साइन पूर्णांकाचे प्रतिनिधित्व आणि फ्लोटिंग पॉइंट नंबर, तसेच संबंधित एरिथमेटिक ऑपरेशन्स, कसे डिजिटल होते हे स्पष्ट करते. संगणक या क्रमांकांसह विविध ऑपरेशन्स सेटयित आणि कार्यान्वित करतात.

डेटा ट्रान्समिशन दरम्यान नॉईसच्या हस्तक्षेपामुळे डेटामध्ये त्रुटी निर्माण होतात. तंत्र त्रुटी ओळखणे देखील स्पष्ट केले आहे. विद्यार्थी एरिथमेटिक लॉजिक शिफ्ट युनिटच्या संरचनेबद्दल शिकतात, जे एरिथमेटिक, लॉजिक आणि शिफ्ट मायक्रोऑपरेशन्स समजून घेण्यासाठी आवश्यक आहे. या युनिटमुळे डिजिटल संगणकाचे हार्डवेअर घटक ओळखणे आणि ते एकत्र कसे कार्य करतात हे समजून घेणे सोपे होते. हे एकक या पुस्तकातील पुढील घटक समजून घेण्यासाठी पाया प्रदान करते.

पूर्व-आवश्यकता

गणित: पूर्णांक आणि फ्लोटिंग पॉइंट क्रमांकांसह एरिथमेटिक ऑपरेशन्स (दहावी वर्ग) डिजिटल इलेक्ट्रॉनिक्स: नंबर सिस्टम्स आणि डिजिटल लॉजिक गेट्स (पॉलिटेक्निक इंजिनिअरिंग)

युनिटचे निष्पत्ती (UNIT OUTCOMES)

या युनिटच्या निष्पत्तीची यादी खालीलप्रमाणे आहे.

U1-O1: संगणकाच्या कार्यात्मक एककांच्या मूलभूत संरचनेचे वर्णन करणे.

U1-O2: इंटेजर आणि फ्लोटिंग पॉइंट क्रमांक प्रणालीचे वर्णन करणे.

U1-O3: डेटा ट्रान्समिशन मुळे झालेल्या बिटस्ट्रीम त्रुटी ओळखण्यासाठी एरॉर डिटेक्शन कोड लागू करणे.

U1-O4: बस आणि मेमोरी ट्रान्सफर समजावून घेणे.

U1-O5: मायक्रो-ऑपरेशन्सच्या अंमलबजावणी साठी अरीथमेटिक आणि लॉजिक शिफ्ट युनिट भूमिका समजावून घेणे.

युनिट -1 निष्पत्ती	निष्पत्ती सह अपेक्षित मापिंग				
	1- कमकुवत मापिंग, 2- मध्यम मापिंग, 3- मजबूत मापिंग				
	CO-1	CO-2	CO-3	CO-4	CO-5
U1-01	3	3	3	2	2
U1-02	3	3	2	3	2
U1-03	2	1	3	1	2
U1-04	3	3	2	1	1
U1-05	3	3	3	1	3

1.1 डिजिटल संगणक

डिजिटल संगणक हे रजिस्टरमध्ये साठवलेल्या माहितीवर केलेल्या सूक्ष्म कार्यांच्या मालिकेद्वारे आयोजित केले जातात. डिजिटल संगणक विविध प्रकारची सूक्ष्म कार्ये करण्यास सक्षम आहेत आणि विशिष्ट क्रमाने कार्य करण्यासाठी प्रोग्राम देखील केले जाऊ शकतात. संगणक वापरकर्ता प्रोग्रामद्वारे क्रियाकलाप नियंत्रित करू शकतो.

कॉम्प्युटर प्रोग्रॅम इंस्ट्रक्शनच्या क्रमाने ऑपरेशन्स, ऑपरेंड्स आणि प्रोसेसिंग ऑर्डर निर्दिष्ट करतो. संगणकाच्या इंस्ट्रक्शन आणि डेटा बायनरी अंक म्हणून एन्कोड केले जातात, 0 किंवा 1, ज्याला बिट्स म्हणतात. डेटा प्रोसेसिंग कार्य बदलण्यासाठी, एकतर नवीन इंस्ट्रक्शनसह एक नवीन प्रोग्रॅम किंवा नवीन डेटासह समान इंस्ट्रक्शन दिल्या जाऊ शकतात.

संगणकासाठी सूक्ष्म ऑपरेशन्सची मालिका निर्दिष्ट करण्यासाठी बायनरी अंकांचा एक भाग म्हणून संगणक डारेक्टिव एन्कोड केला जातो. इंस्ट्रक्शन कोड तसेच डेटा मेमोरी मध्ये सेटयित केली जाते. प्रत्येक इंस्ट्रक्शन मेमोरीतून वाचली जाते आणि प्रोसेसर रजिस्टरमध्ये संग्रहित केली जाते. कंट्रोलर इंस्ट्रक्शनच्या बायनरी कोडचा अर्थ लावतो आणि सूक्ष्म क्रियांची मालिका जारी करून त्याची अंमलबजावणी करतो. [1]. प्रत्येक संगणकाच्या स्वतःच्या इंस्ट्रक्शन सेट असतात. इंस्ट्रक्शन सेटयित आणि एक्सिक्युशन आणण्याच्या सामान्य उद्देशाच्या संगणकाच्या क्षमतेला सेटयित प्रोग्रॅम (stored program) संकल्पना म्हणून ओळखले जाते. विविध संगणकाचे कार्यात्मक घटक दिलेली कार्ये सहकार्याने पार पाडतात.

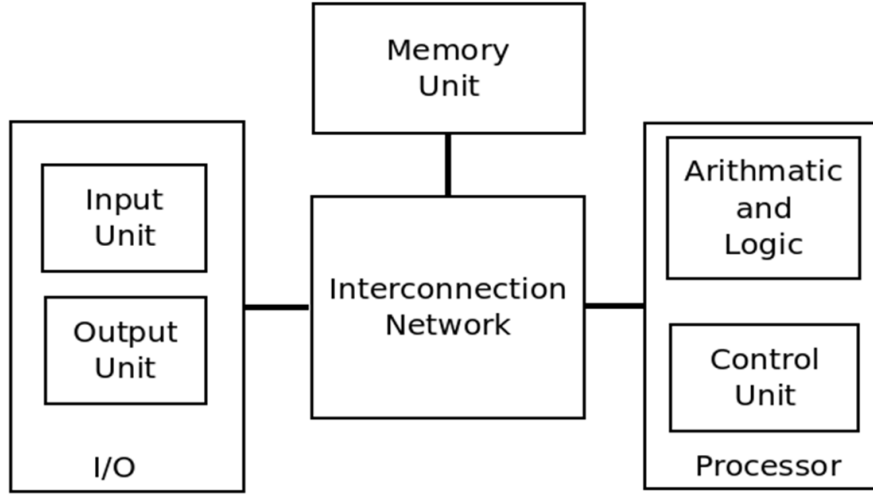
1.2 संगणक कार्यात्मक युनिट्स

आकृती 1.1 मध्ये दर्शविल्याप्रमाणे संगणक इनपुट युनिट, मेमोरी युनिट, एरिथमेटिक लॉजिक युनिट (ALU), आउटपुट युनिट आणि कंट्रोल युनिटने बनलेला आहे. इनपुट युनिट कीबोर्ड किंवा डिजिटल कम्युनिकेशन लाईन्समधून कोडित माहिती स्वीकारते. प्राप्त केलेला डेटा ALU द्वारे नंतरच्या वापरासाठी किंवा तात्काळ प्रोसीजरसाठी मेमोरीमध्ये संग्रहित केला जातो. मेमोरी मध्ये साठवलेले प्रोग्रॅम प्रोसीजरच्या स्टेप निर्दिष्ट करतात. आउटपुट युनिट निकाल बाहेर पाठवते. कंट्रोल युनिट सर्व क्रियांचे समन्वय करतात. हे कार्यात्मक घटक आंतरजोडणी नेटवर्कद्वारे माहितीची देवाणघेवाण करू शकतात.

प्रोसेसर/सीपीयू एरिथमेटिक लॉजिक युनिट्स (ALU) आणि कंट्रोल युनिट्स (CU) नि बनलेले आहे [2]. आय/ओ (इनपुट/आउटपुट) इनपुट आणि आउटपुट दोन्ही उपकरणांचा संदर्भ देतात. संगणक वापरकर्ता स्पष्ट आदेश (एकतर इंस्ट्रक्शन किंवा डेटा) पाठवीतो.



स्कॅन करा
इंटरएट आणि त्यांचे प्रकार समजून
घेण्यासाठी



आकृती. 1.1 संगणक कार्यात्मक एकक

- मेमोरीमध्ये आवश्यक प्रोग्राम लोड करून प्रोसेसरकडे माहितीचे ट्रान्स्फर नियंत्रित करतात.
- आय/ओ उपकरणांद्वारे वापरकर्ता आणि प्रोसेसर यांच्यातील परस्परसंवाद सक्षम करतात.
- प्रोसेसरद्वारे अंमलात आणल्या जाणाऱ्या एएलयू ऑपरेशन्स निर्दिष्ट करणे.

प्रोसेसर प्रोग्रामच्या इंस्ट्रक्शनचे पालन करतो आणि आवश्यक कार्ये अनुक्रमिक पद्धतीने पूर्ण करतात. वापरकर्त्याद्वारे किंवा त्याच्याशी जोडलेल्या आय/ओ उपकरणांद्वारे संभाव्य बाह्य व्यत्यय वगळता, संगणक संग्रहित प्रोग्रामद्वारे नियंत्रित केला जातो. संख्या (number) आणि अक्षरे (character) डेटा हा इंस्ट्रक्शन द्वारे ऑपरेंड म्हणून वापरली जातात. [3]

1.2.1 इनपुट युनिट्स

इनपुट युनिट्स संगणकांना डेटा प्राप्त करण्यास परवानगी देतात. कीबोर्ड हे मूलभूत इनपुट उपकरण आहे. जेव्हा एखादी की प्रेस केली जातात, तेव्हा की शी संबंधित अक्षर किंवा अंक त्याच्या संबंधित बायनरी अंकात रूपांतरित केला जातो आणि प्रोसेसरकडे पाठवीला जातो.

मानवी संगणकाच्या परस्परसंवादासाठी विविध प्रकारची इनपुट उपकरणे आहेत जी आकृती 1.2 मध्ये दर्शविल्याप्रमाणे वापरली जाऊ शकतात. OCR (ऑप्टिकल कॅरेक्टर रिकग्निशन) माऊस, बारकोड रीडर, जॉयस्टिक, टच पॅनेल, टचपॅड, एमआयसीआर (मॅग्नेटिक इंक कॅरेक्टर रिकग्निशन) ट्रॅकबॉल आणि स्कॅनर ही या उपकरणांची काही उदाहरणे आहेत.

ग्राफिकल इनपुट उपकरणे म्हणून काम करण्यासाठी प्रदर्शनाच्या संयोगाने हे वारंवार वापरले जातात. ऑडिओ इनपुट मायक्रोफोनचा वापर करून कॅप्चर केले जाऊ शकते आणि एकदा ते पूर्ण झाल्यावर त्याचे नमुने घेतले जातात आणि डिजिटल कोडमध्ये रूपांतरित केले जातात जेणेकरून ते संग्रहित आणि प्रक्रिया केले जाऊ शकते.



आकृती. 1.2: इनपुट उपकरणे

त्याचप्रमाणे, व्हिडिओ इनपुट कॅमेऱ्यांच्या वापरा द्वारे टिपले जाऊ शकते. संगणक इंटरनेटच्या डिजिटल संप्रेषण सुविधेचा वापर करून केवळ इतर संगणकांकडूनच नव्हे तर डेटाबेस सर्व्हरवरूनही इनपुट घेऊ शकतात.

1.2.2 मेमोरी: मेमोरी युनिट डेटा आणि प्रोग्राम्स साठवतात. म्हणून प्राथमिक आणि दुय्यम स्टोरेज ओळखल्या जाणाऱ्या दोन स्टोरेज श्रेणी आहेत.

प्राथमिक मेमोरी

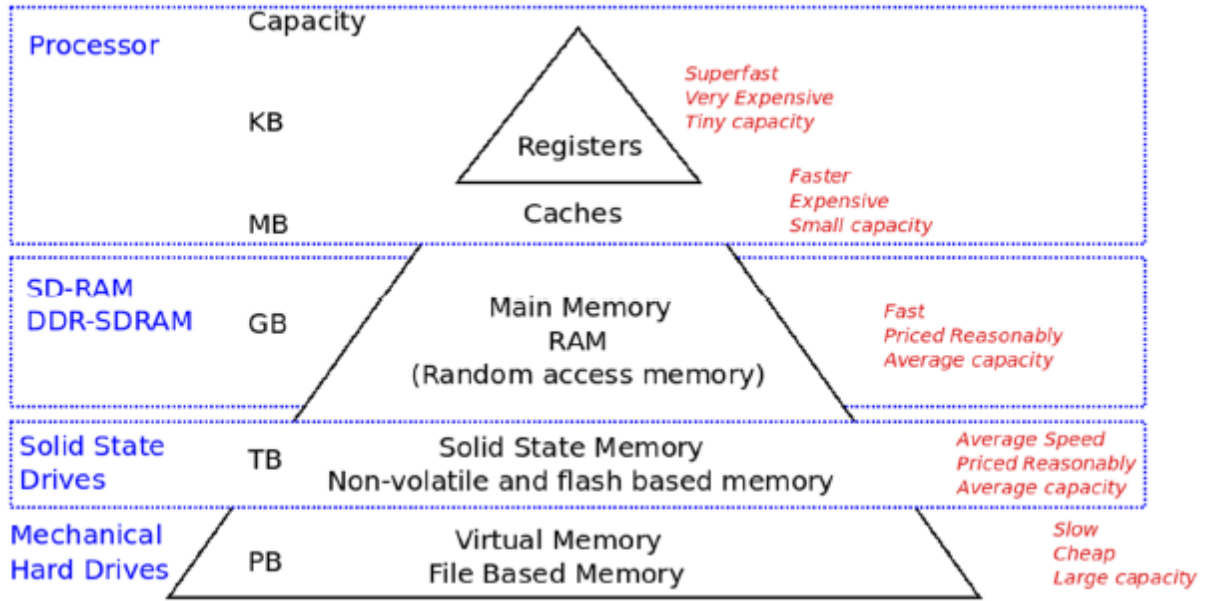
प्राथमिक मेमोरी (primary memory) म्हणजे प्राथमिक भौतिक मेमोरी होय. डेटा आणि प्रोग्रामिंग या दोन्हींच्या सेव्ह करण्यासाठी मेमोरी वापर केला जातो. त्यात असलेल्या असंख्य सेमीकंडक्टर स्टोरेज सेल्सपैकी प्रत्येकामध्ये थोडीशी माहिती साठवली जाऊ शकते. जे मेमोरी सेल पूर्वनिर्धारित आकाराच्या गटांमध्ये आयोजित केल्या जातात ज्यांना वर्ड (वर्ड) म्हणतात. मेमोरीची रचना एकाच ऑपरेशनमध्ये एकच वर्ड संचयित करण्यासाठी किंवा पुनर्प्राप्त करण्यासाठी केली जाते. वर्डची (वर्ड) लांबी म्हणजे प्रत्येक संगणकाच्या वर्डतील बिट्सची संख्या, जी सामान्यतः 16,32 किंवा 64 बिट्स असतात. मेमोरीतील कोणत्याही वर्डपर्यंत प्रवेश सुलभ करण्यासाठी मेमोरीतील प्रत्येक वर्डच्या स्थानासाठी एक अद्वितीय ऍड्रेस नियुक्त केला जातो.



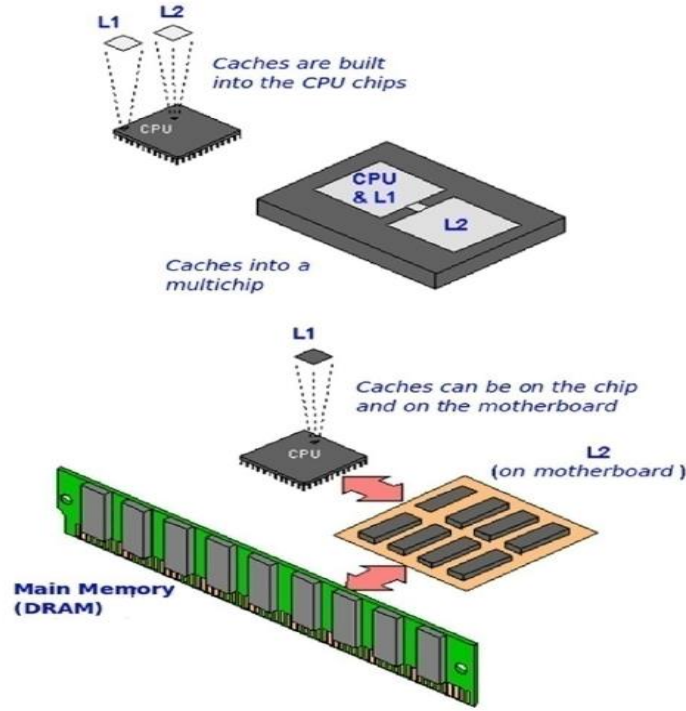
स्कॅन करा
इन पुटयुनिट्सचा उद्देश
समजून घेण्यासाठी

मेमोरी ऍड्रेस (memory address) हा नेहमी शून्यापासून सुरू होतो. एखाद्या विशिष्ट वर्डमध्ये प्रवेश करण्यासाठी, तुम्ही प्रथम त्याचा ऍड्रेस ओळखणे आवश्यक आहे आणि नंतर वर्ड संचयित करण्याची किंवा पुनर्प्राप्त करण्याची प्रक्रिया सुरू करण्यासाठी मेमोरीला कंट्रोल इंस्ट्रक्शन जारी करणे आवश्यक असतात. मेमोरी CPU नियंत्रणाखाली इंस्ट्रक्शन आणि डेटा संचयित करते आणि पुनर्प्राप्त करतात. ऍड्रेसिंग केल्यानंतर, प्रत्येक वर्ड

पूर्वनिर्धारित वेळेत पुनर्प्राप्त केला जाऊ शकतो. 'रँडम-एँक्सेस मेमोरी' (RAM) ही संज्ञा या प्राथमिक मेमोरीचा (primary memory) संदर्भ देतात. मेमोरीतून एक वर्ड पुनर्प्राप्त करण्यासाठी लागणारा वेळेला मेमोरी एँक्सेस टाइम (मेमोरी एँक्सेस टाइम) म्हणतात. या क्षणी वर्डची पोजिशन इरिलेवन्ट असतात. आकृती 1.3 मध्ये दर्शविल्याप्रमाणे, समकालीन रँम उपकरणांमध्ये काही नॅनोसेकंद (ns) ते 100 ns पर्यंत विलंब असतात. मेमोरी च्या भिन् भिन् प्रकारामुळे डेटा प्रवेशाच्या गतीवर परिणाम होऊ शकतो. मेमोरी क्षमता KB (Kilobyte) ते GB (Gigabyte) पर्यंत असते. रजिस्टर हा मेमोरीचा सर्वात लहान आणि वेगवान घटक आहे. याउलट, दुय्यम स्टोरेज हा सर्वात मोठा आणि सर्वात संथ मेमोरी प्रकार आहे. प्रोसेसरचे लॉजिक सामान्यतः मुख्य मेमोरी एँक्सेस वेळेपेक्षा वेगवान असते. प्रामुख्याने, प्रोसेसिंगची गती मुख्य मेमोरीच्या गती मुळे मर्यादित असते.



आकृती. 1.3: मेमोरी पदानुक्रम: रजिस्टर, कॅशे आणि मुख्य मेमोरी ही अस्थिर मेमोरी आणि सॉलिड स्टेट ड्राइव्ह आहेत, यांत्रिक हार्ड ड्राइव्ह ही अस्थिर नसलेली मेमोरी आहेत.



आकृती. 1.4: कॅशे मेमोरीचे स्थाननिर्धारण

मुख्य मेमोरीमध्ये प्रवेश करण्याची वेळ आणि प्रोसेसरच्या अंमलबजावणीची गती यांच्यातील गती विसंगतीवर कॅशे मेमोरी द्वारे मात करू शकते. आकृती 1.4 मध्ये दर्शविल्याप्रमाणे प्रोसेसर आणि मुख्य मेमोरी दरम्यान एक लहान, वेगवान कॅशे स्थित असते.

आकृती 1.4. मध्ये दर्शविल्याप्रमाणे डेटा अक्सेस वेळ स्पीड-अप करण्यासाठी प्रोसेसर आणि रॅम च्या दरम्यान एक संक्षिप्त, वेगवान कॅशे मेमोरी ठेवली जाते. प्रोसेसिंग करताना वारंवार वापरलेला डेटा आणि कोड कॅशे मेमोरी मध्ये स्टोर केली जाते.

आधुनिक प्रोसेसरमध्ये कॅशे मेमोरीचे दोन किंवा तीन स्तर असतात. L1 कॅशे प्रोसेसर जवळील चिपवर ठेवला जातो, तर L2 कॅशे प्रोसेसर आणि मेमोरीच्या दरम्यान कुठेही असू शकतो. वैकल्पिकरित्या, दोन्ही L1, L2 कॅशे एकाच चिपवर एकत्रित केले जाऊ शकतात. चिप डिझाइनर चिपची निवड ही, चिपची कार्यक्षमता (performance), शक्ती (power) आणि किंमत (cost) यावर आधारित करतात. L1 कॅशेचा ऍक्सेस टाइम प्रोसेसर च्या लॉजिक क्लॉकसायकल शी जवळजवळ तुलनात्मक आहे.



आकृती. 1.5: दुय्यम स्टोरेज

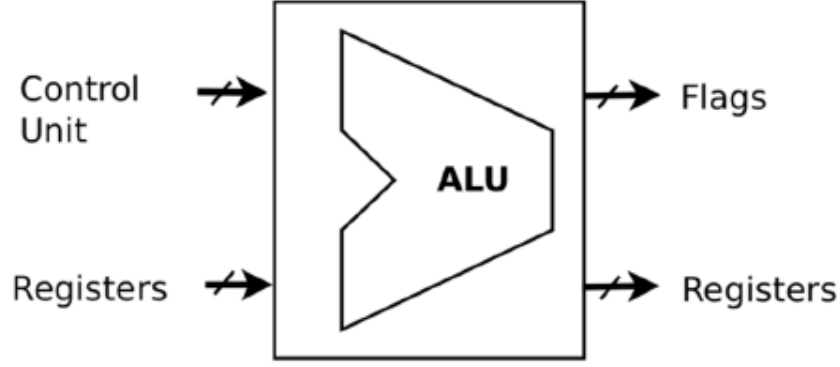
- **दुय्यम मेमोरी**

पॉवर बंद केल्यावर, प्राथमिक मेमोरी तील सर्व संग्रहित डेटा नाहीसा होतो, तर दुय्यम स्टोरेज डेटा आणि प्रोग्राम्स कायमस्वरूपी संग्रहित करते. प्राथमिक मेमोरीपेक्षा दुय्यम मेमोरी ला जास्त ऍक्सेस टाइम्स लागतो. SSD (सॉलिड स्टेट ड्राइव्ह), पेनड्राईव्ह (USB फ्लॅश ड्राइव्ह), SD (सुरक्षित डिजिटल) कार्ड, फ्लॉपी डिस्कट, माग्नेटिक डिस्कस, ऑप्टिकल डिस्कस (डीव्हीडी आणि CD), आणि हार्ड डिस्क ड्राइव्ह असे अनेक दुय्यम स्टोरेज पर्याय उपलब्ध आहेत, जे आकृती 1.5 मध्ये दर्शवीले आहेत.

1.2.3 अरीथमेटिक लॉजिक युनिट

अरीथमेटिक लॉजिक युनिट (ALU) बेरीज, वजाबाकी, गुणाकार, भागाकार आणि संख्यांची तुलना यासारख्या एरिथमेटिक आणि लॉजिक क्रिया कार्यान्वित करतात. जेव्हा प्रोसेसरला आवश्यक ऑपरेंड मिळतात, तेव्हा ते प्रोसेसर रजिस्टरमध्ये साठवले जातात. प्रत्येक रजिस्टरमध्ये डेटाचा एकच वर्ड असू शकतो. त्यानंतर एएलयू त्याची अंमलबजावणी सुरू करते.

दोन संख्यांची जोडणीचा विचार करा; हे ऑपरेशन फक्त मेमोरीतून संख्या पुनर्प्राप्त करून केले जाऊ शकते. नंतर कंट्रोल युनिट एएलयू ला संख्या जोडण्यासाठी सूचित करतात. ही बेरीज मेमोरीमध्ये साठवली जातात किंवा प्रोसेसर च्या तात्काळ वापरासाठी सामान्य रजिस्टर मध्ये साठवू शकते.



आकृती. 1.6: एरिथमेटिक लॉजिक एकक (ALU)

1.2.4 कंट्रोल युनिट

कंट्रोल युनिट प्रोसेसरच्या कार्यावर देखरेख ठेवतात. कंट्रोल युनिट मुख्य मेमोरीतून प्रोग्राम पुनर्प्राप्त करतात आणि इंस्ट्रक्शन रजिस्टर (IR) मध्ये स्टोर करतात. कंट्रोल सिग्नल ALU ला तीच इंस्ट्रक्शन अंमलात आणण्याची इंस्ट्रक्शन देतात. अंमलबजावणी पूर्ण झाल्यावर, कंट्रोल युनिट वापरकर्त्याला परिणाम प्रदर्शित करण्यासाठी आउटपुट एककाला सूचित करते. मेमोरी, एएलयू आणि आय/ओ युनिट च्या कृतींचे समन्वय कंट्रोल युनिट साधते. कंट्रोल युनिट चे कार्ये खालीलप्रमाणे वर्गीकृत केली जाऊ शकतात:

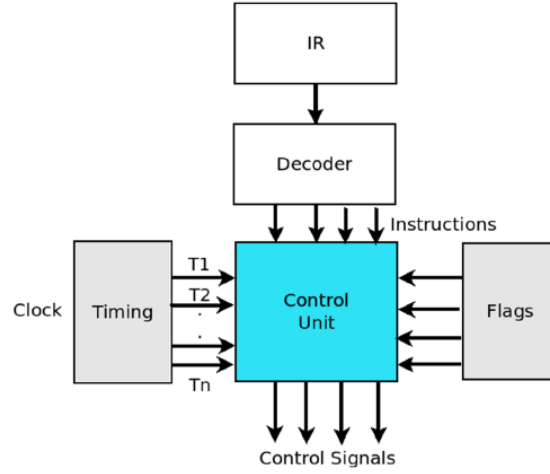


स्कॅन करा

इतर कार्यात्मक घटकांसह
नियंत्रण घटकांचा परस्परसंवाद
समजून घेण्यासाठी

- 1) संगणकाच्या प्रोसेसिंग युनिटमध्ये माहिती कशी प्रवेश करते, बाहेर पडते आणि ट्रान्सफर करणे हे याचे प्राथमिक कार्य आहे.
- 2) कोणती उपकरणे आणि प्रोसेसिंग नियंत्रित करणे आवश्यक आहे आणि त्यांना कसे सक्रिय करावे, तसेच मुख्य मेमोरीतून इंस्ट्रक्शन पुनर्प्राप्त करणे आणि डीकोड करणे ठरवते.
- 3) चिपमधील डेटा प्रोसेसिंग कंट्रोल मध्ये ठेवतात.
- 4) बाहेरून इंस्ट्रक्शन किंवा आदेश घेतात आणि त्यांचे कंट्रोल सिग्नल च्या संचामध्ये रूपांतर करतात.
- 5) प्रोसेसरच्या अनेक एक्झिक्युशन युनिट्सचे (एएलयू, डेटा बफर आणि रजिस्टर) व्यवस्थापन करतात.
- 6) कंट्रोल युनिटचे टाइमर सिग्नल प्रोसेसर आणि मेमोरीमधील माहितीच्या देवाणघेवाणीवर लक्ष ठेवतात.
- 7) इंस्ट्रक्शन आणणे, डीकोडिंग करणे, अंमलात आणणे आणि परिणाम स्टोर करणे यां संबंधित कार्ये करते.

कंट्रोल सर्किटरी संपूर्ण संगणकावर भौतिकरित्या पसरलेली असते. सिग्नल कंट्रोल लाइनच्या संचाद्वारे वाहून नेले जातात (wires). आकृती 1.7 मध्ये दर्शविल्याप्रमाणे कंट्रोल सर्किटमधून वेळेच्या संकेतांद्वारे सर्व युनिट्सच्या घटना समक्रमित केल्या जातात.



आकृती. 1.7: कंट्रोल युनिट इंस्ट्रक्शन डीकोड करते आणि मास्टर क्लॉकद्वारे कंट्रोल सिग्नल तयार करतेइव्हेंट सिंक्रोनाइझ करा (IR: इंस्ट्रक्शन रजिस्टर)

कंट्रोल युनिट सिग्नलचा एक क्रम तयार करतो जो मास्टर क्लॉक आणि संबंधित प्रक्रिया युनिट चालवते. कंट्रोल युनिटच्या मुख्य कार्य इंस्ट्रक्शन इंटरप्रेट करणे आणि त्यांची अंमलबजावणी आयोजित करणे आहे. कंट्रोल युनिट एएलयू ला कोणती कार्ये पार पाडायची आणि त्या कार्यासाठी कोणत्या वेळेचे मापदंड (टायमिंग पॅरामीटर) वापरायचे याबद्दल इंस्ट्रक्शन पाठवतात. प्रोसेसर, मेमोरी आणि इतर घटकांदरम्यान डेटाच्या प्रवासाच्या गतीचे नियमन कंट्रोल युनिटचे टायमिंग सिग्नल करतात.

प्रोग्राम काउंटर (PC) रजिस्टर क्रमाने सध्याच्या इंस्ट्रक्शन क्रमांकाचा मागोवा ठेवतात. चालू असलेली वर्तमान इंस्ट्रक्शन IR रजिस्टर मध्ये स्टोर केली जातात. कंट्रोल युनिट IR मधील माहिती टायमिंग सिग्नल निर्माण करण्यासाठी वापरली जातात.

1.2.5 इंटरकनेक्शन

प्रोसेसर, मेमोरी आणि आय/ओ उपकरणांमध्ये डेटा ट्रान्सफर करणारी प्रोग्रामेबल प्रणाली इंटरकनेक्शन नेटवर्क म्हणून ओळखली जातात. आकृती 1.8 मध्ये दर्शविल्याप्रमाणे, मल्टीप्रोसेसर प्रणाली बसद्वारे इतर प्रोसेसरशी संवाद साधतात.

बसमुळे P0, P1, P2, P3 आणि P4 या पाच प्रोसेसरना एकमेकांशी संवाद साधता येतो. दुसरीकडे, बसेसची बँडविड्थ मर्यादित असते. रिंग्ज, मेश, ट्री, हायपरक्यूब आणि क्रॉसबार यासह इतर आंतरजोडणी नेटवर्कचा वापर अतिरिक्त प्रोसेसर जोडण्यासाठी केला जातो. आकृती 1.8 मध्ये या परस्पर जोडण्यांची रचना दर्शविली आहे.

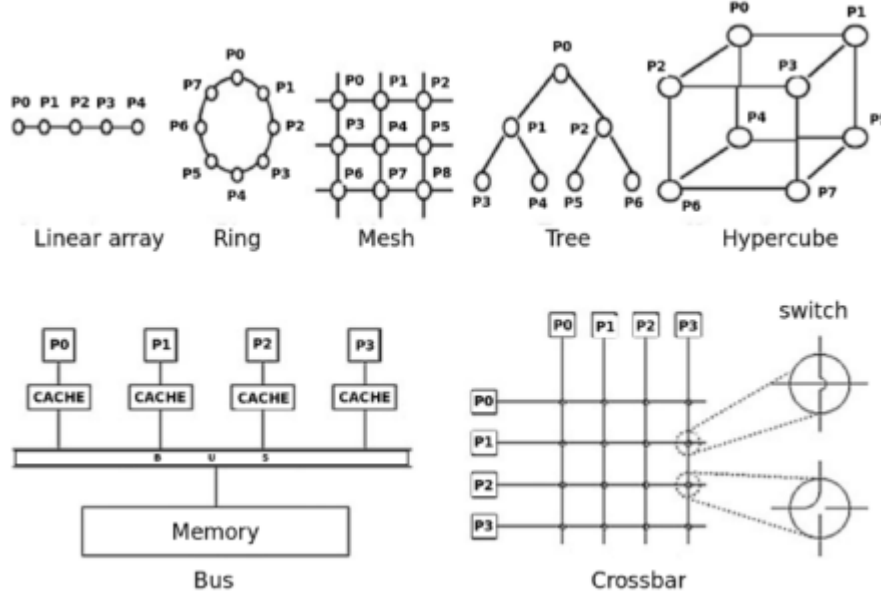


स्कॅनकरा
आंतरजोडणीच्या तपशीलवार
रचनेसाठी



स्कॅनकरा
विविध उत्पादन साधनांचा
उद्देश समजून घेण्यासाठी

एक चांगले प्रकारे डिझाइन केलेले इंटरकनेक्शन नेटवर्क मर्यादित कम्युनिकेशन संसाधनांचा सर्वोत्तम वापर करते. उच्च बँडविड्थ आणि कमी विलंब (लो ल्याटेन्सी) राखताना अनेक हार्डवेअर घटकांमध्ये संवाद साधणे शक्य आहे. बस व्यतिरिक्त पारंपारिक आंतरजोडणीमध्ये लिनिअर अरे, रिंग आणि क्रॉसबार यांचा समावेश होतो. आकृती 1.8 मध्ये दर्शविल्याप्रमाणे मेश, ट्री आणि हायपरक्यूब इंटरकनेक्ट्स अधिक प्रोसेसर कनेक्ट करण्यासाठी वापरले जातात. साध्या रचना तत्वांमुळे आंतरजोडणी नेटवर्क अनेकदा त्यांच्या विशिष्ट अनुप्रयोगाच्या गरजा भागविण्यासाठी सरळ असतात.



आकृती. 1.8: इंटरकनेक्शन नेटवर्क

1.2.6 आउटपुट युनिट

आउटपुट युनिट प्रोसीजर चे परिणाम बाहेरील जगाला पाठवते. मॉनिटर, प्रिंटर, स्पीकर, हेडफोन आणि प्रोजेक्टर यासह आउटपुट डिव्हाइसेसची काही उदाहरणे आकृती. 1.9 मध्ये प्रदर्शित केली गेली आहे. या आउटपुट साधनांपैकी मुद्रक (प्रिंटर) ही यांत्रिक उपकरणे आहेत. ते इलेक्ट्रॉनिक घटकांपासून तयार केलेल्या प्रोसेसरपेक्षा हळू असतात. बहुसंख्य प्रिंटर एकतर शाईच्या प्रवाहाचा वापर करतात किंवा फोटोकॉपी करणे, जसे की लेसर प्रिंटर. ही प्रिंटर प्रति मिनिट 20 पृष्ठे किंवा त्याहून अधिक वेगाने छापू शकतात.



आकृती. 1.9: आउटपुट युनिट

ग्राफिक डिस्प्लेसारखे काही घटक, मजकूर आणि चित्रे दोन्ही दाखवू शकतात आणि टचस्क्रीन ठेवून युजर कडून माहिती घेऊ शकतात. जे युनिट्स इनपुट आणि आउटपुट असे दोन्ही कार्य करतात, त्यांना इनपुट/आउटपुट (I/O) युनिट्स म्हणतात.

1.3 वॉन-न्यूमन आर्किटेक्चर

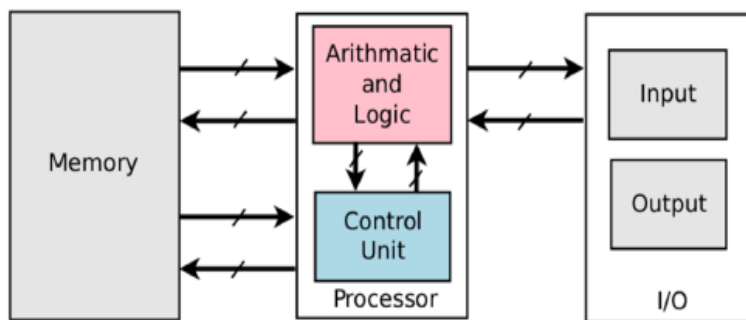
प्रिन्सटनमध्ये, व्हॉन न्यूमन आणि त्याच्या सहकाऱ्यांनी 1946 मध्ये "संचयित प्रोग्राम संगणक (स्टोर प्रोग्राम कंप्युटर)" चा शोध लावला. वॉन न्यूमनची संचयित-प्रोग्राम (स्टोर प्रोग्राम) संकल्पना हा आधुनिक डिजिटल संगणकांचा पाया आहे. आकृती 1.10 मध्ये दर्शविल्याप्रमाणे वॉन न्यूमन आर्किटेक्चरमध्ये कंट्रोल युनिट, अरीथमेटिक लॉजिक युनिट (एएलयू), मेमोरी युनिट, रजिस्टर आणि इनपुट/आउटपुट युनिट यांचा समावेश आहेत.

वॉन न्यूमन आर्किटेक्चरमध्ये प्रोग्राम आणि डेटा संचयित करण्यासाठी एकच सामायिक (shared) मेमोरी वापरली जाते आणि प्रोसेसरद्वारे मेमोरी ऍक्सेस करीन्या करिता एकच बस वापरली जाते. प्रोसेसर अनुक्रमिक पद्धतीने इंस्ट्रक्शन पुनर्प्राप्त आणि अंमलात आणतो. व्हॉन-न्यूमन आर्किटेक्चर डिझाईन च्या अडथळ्यांवर मात करण्यासाठी हार्वर्ड आर्किटेक्चर ची रचना करण्यात आली होती.



स्कॅन करा

व्हॉन-न्यूमन आणि हार्वर्ड
आर्किटेक्चरमधील फरक समजून
घेण्यासाठी



आकृती. 1.10: वॉन-न्यूमन आर्किटेक्चर

हार्वर्ड आर्किटेक्चर प्रोग्राम कोड/ इंस्ट्रक्शन आणि माहितीसाठी दोन स्वतंत्र मेमोरी वापरते. प्रत्येक मेमोरी विभागासाठी स्वतंत्र समर्पित बस वापरून प्रोसेसर एकाच वेळी प्रोग्राम कोड आणि डेटा वापरू शकतात. प्रोसेसर एकाच वेळी डेटा आणि इंस्ट्रक्शन आणू शकतो, ज्यामुळे एक्झिक्युशन चा वेग वाढतो.

सुधारित हार्वर्ड रचना सामान्यतः समकालीन प्रोसेसर मध्ये वापरली जाते. डेटा आणि इंस्ट्रक्शन चिपवरील दोन वेगवेगळ्या कॅशमध्ये संग्रहित केल्या जातात. X86 आणि ARM प्रोसेसर या दोन्हीमध्ये हे वैशिष्ट्य आहे.

1.4 बस संरचना

बायनरी अंक एका रजिस्टर/युनिटमधून दुसऱ्या युनिट्स/रजिस्टरमध्ये नेटवर्कद्वारे ट्रान्सफर केले जातात अशा भौतिक तारांना बस म्हणतात. बस सर्व प्राथमिक अंतर्गत घटकांना सीपीयू/प्रोसेसर आणि मेमोरीशी जोडतात आणि ते त्यांच्यामध्ये डेटा ट्रान्सफर करतात. एकेरी (single) बस, दुहेरी (double) बस आणि एकाधिक (multiple) बस ही सर्वात सामान्य संरचना आहेत.



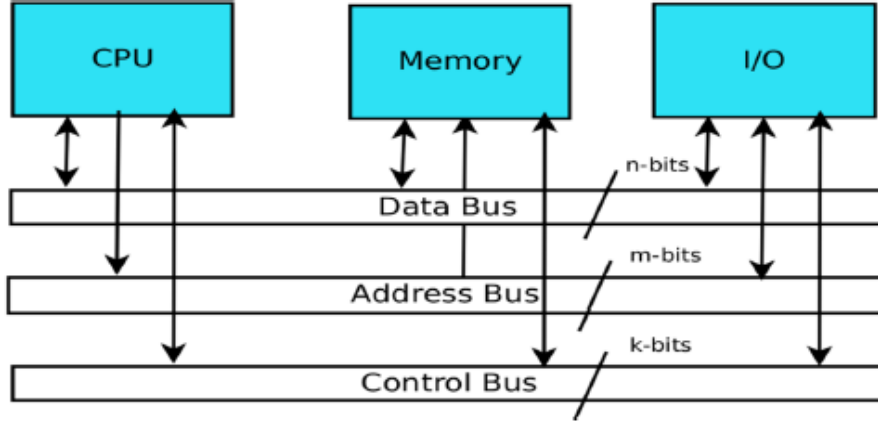
स्कॅन करा
वेगवेगळ्या बस संरचनांची
तुलना करण्यासाठी

सिंगल-बस प्रणालीमध्ये, सर्व युनिट एकाच बसशी जोडल्या जातात, जे कनेक्टिव्हिटीचे एकमेव साधन म्हणून काम करतात. सिंगल-बस इंटरकनेक्ट च्या डिझाईन्स सोप्या आणि किफायतशीर आहेत. तथापि, एकाच वेळी केवळ दोन घटक संवाद साधू शकतात. हे वैशिष्ट्य नेटवर्कचा वेग मर्यादित करते. ही मर्यादा दुहेरी बस संरचनेद्वारे पार केली जाते, ज्या मध्ये दोन बसेसद्वारे आदान प्रदान करते. हार्वर्ड रचनेत, प्रोसेसर एकाच वेळी मेमोरीतून इंस्ट्रक्शन पुनर्प्राप्त करू शकतो आणि मेमोरीमध्ये डेटा लिहू शकतो.

इंस्ट्रक्शन, डेटा, अॅड्रेसिंग सिग्नल इत्यादी माहितीच्या ट्रान्सफरसाठी अनेक अंतर्गत मार्ग प्रदान करण्यासाठी बहुसंख्य व्यावसायिक प्रोसेसरद्वारे अनेक बसेस वापरल्या जातात. आकृती. 1.11 मध्ये CPU (प्रोसेसर), मेमोरी आणि I/O युनिट्सला जोडणारी तीन-बस ची रचना दर्शविते. डेटा बसची रुंदी n बिट्स असते, अॅड्रेस बसची रुंदी m बिट्स असते आणि कंट्रोल बस CPU (प्रोसेसर) मेमोरी आणि I/O उपकरणांदरम्यान इंस्ट्रक्शन k बिट्स वाहून नेऊ शकते. या बसेसना एकत्रितपणे सिस्टम बस म्हणून संबोधले जाते.

या बसेस ट्रान्सफर करतात-

- मेमोरी युनिट, इनपुट/आउटपुट डिव्हाइसेस आणि प्रोसेसर/सीपीयू दरम्यान डेटा बसद्वारे डेटा.
- सर्व जोडलेल्या उपकरणांमधील अॅड्रेसच्या बस द्वारे अॅड्रेस.
- कंट्रोल सिग्नल जे कंट्रोल बसद्वारे पाठवले जातात ते सर्व क्रियाकलापांचे निरीक्षण आणि समन्वय साधण्यासाठी या युनिट्स.



आकृती. 1.11: सिस्टम बसची रचना डेटा बस, ॲड्रेस बस आणि कंट्रोल बस यांनी बनलेली आहे

विशिष्ट डेटा ट्रान्समिशन दरम्यान, कंट्रोल सिग्नल बस कोणते युनिट/रजिस्टर निवड करायचे हे ठरवतात. अनेक बसेसवर डेटा वाहून नेणाऱ्या मल्टिपल बसेस माहिती प्रोसेसिंग ला गती देतात. संगणक विविध उपकरणांशी संवाद साधण्यासाठी मल्टिपल बस व्यतिरिक्त पेरिफेरल बस (आय/ओ बस) स्थानिक बस आणि उच्च-गती बसेस देखील वापरतात.

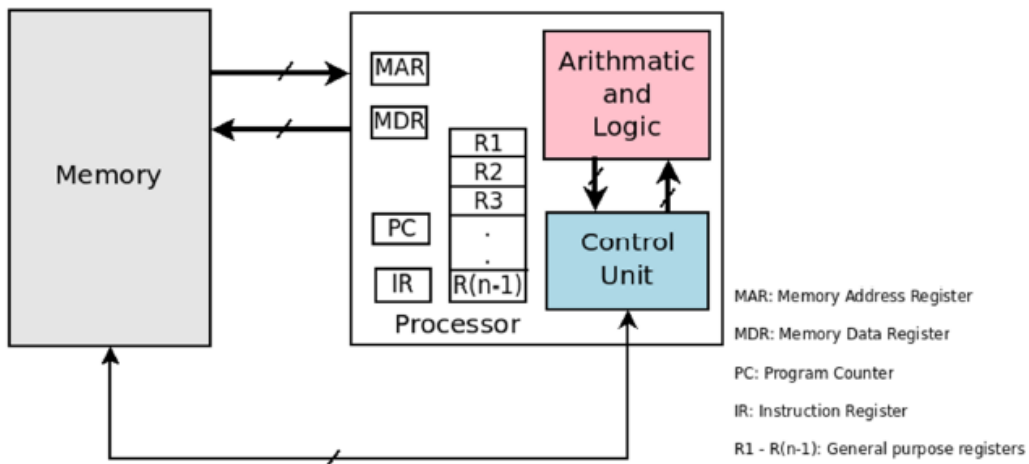
1.5 मूलभूत ऑपरेशनल संकल्पना

संगणक अनेक उपकरणे/एककांच्या प्रयत्नांचे एकत्रीकरण करून अनेक ऑपरेशन साध्य करतात. प्रोसेसरमध्ये एएलयू आणि अनेक वेगवेगळ्या हेतूसाठी वापरल्या जाणार्या कंट्रोल सर्किटरी व्यतिरिक्त अनेक रजिस्टर असतात. आकृती. 1.12 मध्ये प्रोसेसर रजिस्टर, मेमोरी स्ट्रक्चर आणि सिस्टम बस इंटरकनेक्शनची व्यवस्था दर्शविते. प्रोसेसरला आवश्यक असलेली विशिष्ट प्रकारची माहिती तात्पुरती साठवण्यासाठी खालील रजिस्टरचा वापर केला जातो:



स्कॅन करा

बसेसच्या प्रकारांबद्दल तपशीलवार माहितीसाठी



आकृती. 1.12: प्रोसेसर रजिस्टर्स आणि मुख्य मेमोरीची रचना

- प्रोसेसरच्या मेमोरी अॅड्रेस रजिस्टर (MAR) मधून डेटा पुनर्प्राप्त किंवा संग्रहित केला जातो. मेमोरी मध्ये प्रवेश करावयाच्या स्थानाचा अॅड्रेस एमएआर रजिस्टर मध्ये साठवला जातो.
- प्रोग्राम काउंटर (पीसी) पुढील इंस्ट्रक्शन चा मेमोरी अॅड्रेस संग्रहित करतो आणि अंमलात आणतो.
- सध्या अंमलात आणलेल्या इंस्ट्रक्शन इंस्ट्रक्शन रजिस्टर (IR) मध्ये सेव्ह केल्या जातात. एएल्यू इंस्ट्रक्शन चे एक्सिकुशन कंट्रोल युनिट युनिटच्या टायमिंग सिग्नल द्वारे सुरू केली जाते.
- प्रोसेसर रजिस्टर्स ($R_0 - R_{n-1}$) सामान्य हेतू रजिस्टर्स आहेत. ते विविध कारणांसाठी वापरले जातात जसे की तात्पुरते मेमोरीमध्ये लोड/स्टोअर करण्या करिता.



स्कॅन करा
इंटरप्ट आणि इंटरप्ट सर्विस
रुटीन बदल अधिक जाणून
घेण्यासाठी

प्रोग्राम्स एक्सिकुशन पूर्वी, प्रोग्राम्स चे कन्टेन्ट सेकंडरी मेमोरी तून प्राथमरी मेमोरी मध्ये ट्रान्सफर करतात. जेव्हा पीसी (PC) पहिल्या इंस्ट्रक्शनकडे नेव्हिगेट करेल, तेव्हा प्रोग्राम चे एक्सिकुशन सुरू होतात. इंस्ट्रक्शन पुनर्प्राप्त करण्यासाठी वाचन कंट्रोल संकेत (read control signal) मेमोरी कडे पाठविला जातात. मेमोरी मधून मिळवलेला वर्ड (word), जो प्रोग्रॅम ची प्रारंभिक इंस्ट्रक्शन देखील आहे, तो IR रजिस्टर मध्ये स्टोर केला जातो. आता दिलेल्या इंस्ट्रक्शन चा अर्थ जाणून आणि त्याला एक्सिकुशन करणे शक्य आहे. एक्सिकुशन झाल्या नंतर त्याला प्रोसेसर रजिस्टरमध्ये पाठवले जाते. कंट्रोल युनिट मेमोरी मध्ये वर्ड (word) लिहिण्यासाठी अॅड्रेस, वर्ड (word) आणि राईट कंट्रोल सिग्नल मेमोरीला पाठवते.

पुढील इंस्ट्रक्शन डारेक्टिव (एक्सिकुशन) करण्यासाठी इंस्ट्रक्शन च्या एक्सिकुशन दरम्यान PC काउंटर वाढवला जातो. अशा प्रकारे, विद्यमान इंस्ट्रक्शन चे एक्सिकुशन झाल्या नंतर नवीन इंस्ट्रक्शन CPU आणले जाते. संगणकाद्वारे मेमोरी आणि प्रोसेसर, इनपुट उपकरणे आणि आउटपुट उपकरणां दरम्यान डेटा ट्रान्सफर केला जातो. इंस्ट्रक्शन आय/ओ ट्रान्सफरवर देखील कंट्रोल ठेवतात.

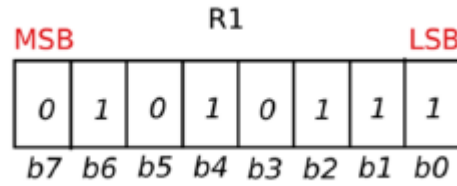
जेव्हा एखाद्या डिव्हाइसला तातडीच्या सेवेची आवश्यकता असते, तेव्हा सामान्य प्रोग्रॅम चे एक्सिकुशन थांबवली जाऊ शकते. जेव्हा एखाद्या उपकरणाला तातडीच्या सेवेची आवश्यकता असते, तेव्हा सामान्य प्रोग्रॅम चे एक्सिकुशन थांबवले जाऊ शकते. उदाहरणार्थ, संगणक-नियंत्रित औद्योगिक प्रोसीजरतील देखरेखीचे उपकरण संभाव्य धोकादायक स्थिती शोधू शकते. तात्काळ प्रतिसाद देण्यासाठी, सध्याच्या प्रोग्रॅम चे एक्सिकुशन थांबवणे आवश्यक आहे. हे साध्य करण्यासाठी, डिव्हाइस प्रोसेसरला इंटरप्ट सिग्नल पाठवते, जी एक सेवा विनंती आहे.

सर्विस विनंती (request) प्रतिसाद म्हणून CPU इंटरप्ट सर्विस रुटीन कार्यान्वित करते. या बदलांमुळे, सर्विस विनंती (request) पूर्ण करण्यापूर्वी CPU ची सध्याची स्थिती मेमोरी मध्ये जतन करणे आवश्यक आहे.

पीसी, सामान्य-उद्देश रजिस्टर (जेनेरल परपज रजिस्टर) आणि कंट्रोल डेटा स्टोर केला जाते. इंटरप्ट-सर्व्हिस रूटीन नंतर, रॅम प्रोसेसरची स्थिती पुनर्संचयित करते जेणेकरून व्यत्यय आलेला प्रोग्राम पुन्हा सुरू होऊ शकतो.

1.6 डेटा प्रतिनिधित्व

संगणक प्रणाली बायनरी संख्या (एकतर 0 किंवा 1) बिट्सची स्ट्रिंग म्हणून संग्रहित करते. वास्तविक जगात, इंटेजर संख्या ही डेसिमल, ऑक्टल आणि हेक्सा-डेसिमल संख्या म्हणून दर्शविल्या जाऊ शकतात. संगणकांमध्ये डेसिमल संख्यांचे बायनरी संख्यांमध्ये रूपांतर केले जाते. बायनरी संख्यांमध्ये, सर्वात डावीकडील बिटला मोस्ट सिग्निफिकन्ट बिट (एमएसबी) म्हणून ओळखले जाते आणि सर्वात उजवीकडील बिटला सर्वात लीस्ट सिग्निफिकन्ट बिट (एलएसबी) म्हणून ओळखले जाते. [7]



आकृती. 1.13: 8-बिट बायनरी नंबरमध्ये MSB आणि LSB बिट्स

उदाहरण आकृतीमध्ये 1.13 मध्ये दर्शविल्या प्रमाणे, बिट 0 हे LSB आहे आणि बिट 7 हे 8-बिट रजिस्टर R1 मध्ये संग्रहित बायनरी नंबरसाठी MSB आहे.

1.6.1 फिक्स्ड पॉइंट पूर्णांक प्रतिनिधित्व

इंटेजर संख्या साइन किंवा अन साइन संख्या असू शकतात. जोपर्यंत साइन संख्या निर्दिष्ट केले जात नाही तोपर्यंत त्या संख्या अन-साइन मानल्या जातात. स्टॅंडर्ड अरीथमेटिक नकारात्मक संख्या नेगेटिव्ह नंबर वजा (-) चिन्हाने आणि पॉजीटीव्ह नंबर अधिक (+) चिन्हाने दर्शविली जाते. यंत्रसामग्रीच्या मर्यादांमुळे, संगणकांनी केवळ 1s आणि 0s वापरून संख्या चिन्हासह प्रत्येक गोष्टीचे प्रतिनिधित्व करणे आवश्यक आहे. चिन्हाचे प्रतिनिधित्व संख्येच्या सर्वात डाव्या स्थानावरील बिटद्वारे केले जाते. सामान्यतः साइन बिट दर्शविताना, पॉजीटीव्ह संख्यांसाठी 0 आणि नेगेटिव्ह संख्यांसाठी 1 असे सेट केले जाते.

साइन नंबर प्रतिनिधित्व (Signed Number Representation)

पॉजीटीव्ह आणि नेगेटिव्ह बायनरी नंबर अनुक्रमे 0 आणि 1 बिट द्वारे दर्शवले जाते. उर्वरित बिट्स खाली सूचीबद्ध केलेल्या संख्या प्रणालींपैकी एकाद्वारे व्यक्त केले जातात:

1. साइन आणि माग्नीटुड
2. 1s कॉम्पलिमेन्ट
3. 2s कॉम्पलिमेन्ट

तक्ता 1.1: पॉजीटीव्ह आणि नेगेटिव्ह बायनरी नंबर, साइन आणि माग्नीटुड, 1s कॉम्पलिमेन्ट आणि 2s कॉम्पलिमेन्ट संख्येचे प्रतिनिधित्व

$b_3b_2b_1b_0$	साइन आणि माग्नीटुड	1's कॉम्पलिमेन्ट	2's कॉम्पलिमेन्ट
0111	+7	+7	+7
0110	+6	+6	+6
0101	+5	+5	+5
0100	+4	+4	+4
0011	+3	+3	+3
0010	+2	+2	+2
0001	+1	+1	+1
0000	+0	+0	+0
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	-7	-1

तक्ता 1.1 मध्ये दर्शविल्याप्रमाणे साइन आणि माग्नीटुड, 1s कॉम्पलिमेन्ट आणि 2s कॉम्पलिमेन्ट चे प्रतिनिधित्व आणि पॉजीटीव्ह संख्यांचे प्रतिनिधित्व समान आहे.

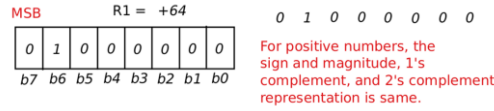
तर, नकारात्मक संख्येच्या चिन्ह आणि परिमाण स्वरूपात परिमाण (संख्येचे बायनरी प्रतिनिधित्व) असते आणि नकारात्मक चिन्हासाठी एमएसबीवर 1 स्थान असते. नकारात्मक संख्या ही एकतर त्याच्या सकारात्मक व्हॅलूचे 1 किंवा 2 चे पूरक असते.

उदाहरण 1.1

+64 आणि -64 संख्या साइन आणि माग्नीटुड, 1s कॉम्पलिमेन्ट आणि 2s कॉम्पलिमेन्ट मध्ये कसे दर्शविले जातात?

उत्तर:

उपाय: आठ बिट्ससह +64 दर्शविण्याचा एकच मार्ग आहे, जरी आकृती 1.14. मध्ये दर्शविल्याप्रमाणे -64 रीप्रेसेन्ट करण्याचे तीन वेगवेगळे मार्ग आहेत. साइन आणि माग्नीटुड 11000000, 1s चे कॉम्पलिमेन्ट 10111111 आणि 2s चे कॉम्पलिमेन्ट 11000000 म्हणून दर्शविले जाते.



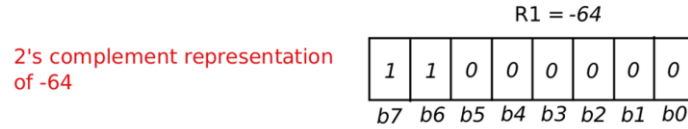
-64 sign and magnitude 1 1 0 0 0 0 0 0

+64 0 1 0 0 0 0 0 0

1's complement 1 0 1 1 1 1 1 1

+0 0 0 0 0 0 0 1

2's complement representation of -64 1 1 0 0 0 0 0 0



आकृती. 1.14: साइन आणि माग्नीटुड, 1s कॉम्पलिमेन्ट आणि 2s कॉम्पलिमेन्ट संख्या प्रणालींमध्ये +64 आणि -64 साइन केलेल्या संख्यांचे प्रतिनिधित्व

-64 चे साइन आणि माग्नीटुड प्रतिनिधित्व फक्त साइन बिटचे कॉम्पलिमेन्ट प्राप्त करून +64 पासून केले जाते. -64 चे 1s कॉम्पलिमेन्ट +64 च्या सर्व बिट्सच्या कॉम्पलिमेन्ट करून साइन बिटसह प्राप्त केले जाते. +64 च्या 2 कॉम्पलिमेन्ट 1 च्या कॉम्पलिमेन्ट मध्ये एक (00000001) जोडून काढले जाते, परिणामी ही संख्या 10111111 असते. साइन आणि माग्नीटुड, 1s कॉम्पलिमेन्ट मध्ये 0 (+0 आणि -0) चे दोन प्रतिनिधित्व आहेत. म्हणून, सूक्ष्म कार्यामध्ये गुंतागुंत टाळण्यासाठी संगणकात नकारात्मक संख्यांचे 2s कॉम्पलिमेन्ट वापरले जातात.

एरिथमेटिक जोडणी मध्ये ओव्हरफ्लो शोध (Overflow Detection in arithmetic additions)

जेव्हा n अंकी संख्यांच्या बेरीजमध्ये n+1 अंक असतात तेव्हा ओव्हरफ्लो होतो. रजिस्टर ची रुंदी डिजिटल संगणकांमध्ये मर्यादित असते. म्हणून, सर्व संगणक ओव्हरफ्लो शोधतील आणि जेव्हा असे होईल तेव्हा ओव्हरफ्लो फ्लॅग सेट केला जाईल.

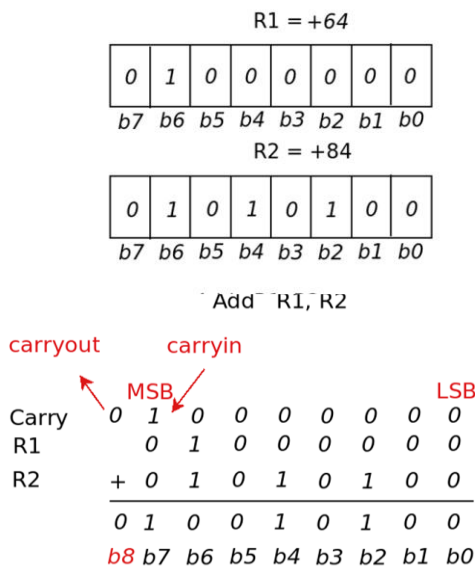
ओव्हरफ्लो डिटेक्शन यंत्रणा ही साइन किंवा अन-साइन संख्येवर अवलंबून असते.

- जेव्हा परिणामांमध्ये एमएसबी कडून 1 अंक जेनरेट होतो तेव्हा अन-साइन क्रमांक जोडल्यावर ओव्हरफ्लो निर्माण होतो.
- साइन केलेल्या संख्यांच्या जोडणीत मूळ संख्यांसह साइन बिट जोडणे समाविष्ट आहे. नकारात्मक संख्यांसाठी 2s कॉम्पलिमेंट वापरले जाते. या प्रकरणात, एंड कॅरी व्हॅल्यू ओव्हरफ्लोकडे डारेक्टिव करत नाही. 2s कॉम्पलिमेंट चे प्रतिनिधित्व n बिट्सना -2^{n-1} ते $+2^{n-1}-1$ पर्यंत व्हॅल्यू सेटयित करण्यास अनुमती देते. उदाहरणार्थ, -8 ते +7 ही 4 बिट्सद्वारे दर्शविली जाऊ शकणारी श्रेणी आहे. जेव्हा ऑपरेशनचा वास्तविक परिणाम दर्शविलेल्या श्रेणीपेक्षा जास्त असतो, तेव्हा ओव्हरफ्लो होतो.
- जेव्हा एक संख्या सकारात्मक आणि दुसरी नकारात्मक असेल तेव्हा ओव्हरफ्लो कंडिशन ची स्थिती उद्भवू शकत नाही, कारण दोन संख्यांची बेरीज नेहमीच दोन संख्यांपैकी मोठ्या संख्येपेक्षा कमी असते.
- जर जोडल्या जाणाऱ्या दोन्ही संख्या सकारात्मक किंवा नकारात्मक असतील, तर ओव्हरफ्लो होऊ शकतो.

उदाहरण 1.2

2s कॉम्पलिमेंट संख्या प्रणालीचा वापर करून R1= +64 आणि R2= +84 या दोन साइन संख्यांची बेरीज करा. रजिस्टरचा सयिज 8-बिट आहे असे समजा.

उपाय: प्रत्येक रजिस्टरचा साईझ 8-बिट आहे. त्यामुळे ते -2^{8-1} ते $+2^{8-1} - 1$ किंवा -128 ते +127 बायनरी संख्या सामावून घेऊ शकतात. दोन्ही दशांश संख्यांची बेरीज केल्यावर, ओव्हरफ्लो आहे की नाही हे आपण थेट तपासू शकतो. $(+64) + (+84) = +148$ ची बेरीज आहे. सकारात्मक दिशेने, ओव्हरफ्लो टाळण्यासाठी परिणाम +127 पेक्षा जास्त असू शकत नाही. जर $+148 > +127$, तर दोन्ही संख्या जोडल्यावर ओव्हरफ्लो होतो.



$$\begin{array}{r}
 \text{R1} \quad 64 \\
 \text{R2} \quad + 84 \\
 \hline
 148
 \end{array}$$

↑
Outside the range of
-128 to +127

The register can store n=8 bits
so $-2^{(8-1)}$ to $2^{(8-1)} - 1$ is the
valid range of numbers

$$\text{Overflow Detection} = \text{XOR}(\text{MSB}_{\text{carryin}}, \text{MSB}_{\text{carryout}})$$

$$= \text{MSB}_{\text{carryin}} \oplus \text{MSB}_{\text{carryout}}$$

$$= 1 \oplus 0$$

$$= 1 \quad \leftarrow \text{The value 1 indicates overflow}$$

आकृती. 1.15: दोन सकारात्मक संख्या + 64 आणि + 84 जोडताना ओव्हरफ्लो डिटेक्शन

आकृती 1.15 मध्ये साइन केलेले बायनरी क्रमांक एरिथमेटिक करताना ओव्हरफ्लो कसे तपासायचे ते दर्शविले आहे. जर कॅरी आउट 8-बिट रजिस्टरच्या क्षमतेपेक्षा जास्त असेल तर याचा अर्थ नेहमीच ओव्हरफ्लो होत नाही. कधीकधी कॅरीआउट कडे दुर्लक्ष केले जाऊ शकते. एमएसबी बिट पोझिशनच्या कॅरी इन आणि कॅरी आउटवर एक्स ऑर (xor) ऑपरेशन करून ओव्हरफ्लो फिक्स्डपणे ओळखला जाऊ शकतो. म्हणून, +64 आणि +84 ची बेरीज ओव्हरफ्लोमुळे चुकीचा परिणाम देते कारण परिणाम 8 बिट्सच्या आत सामावून घेता येत नाही.

साइन बिट पोझिशनमधून कॅरी इन आणि कॅरी आउट (MSB) ओव्हरफ्लो दर्शवते. जर या दोन कॅरीची व्हॅल्यू वेगळी असतील तर ओव्हरफ्लो होईल. जेव्हा दोन्ही कॅरी एक्सक्लुझिव्ह-ओआर. (xor) गेटमध्ये भरल्या जातात, तेव्हा गेटचे आउटपुट 1 च्या बरोबरीचे असते, जे ओव्हरफ्लो दर्शवते.

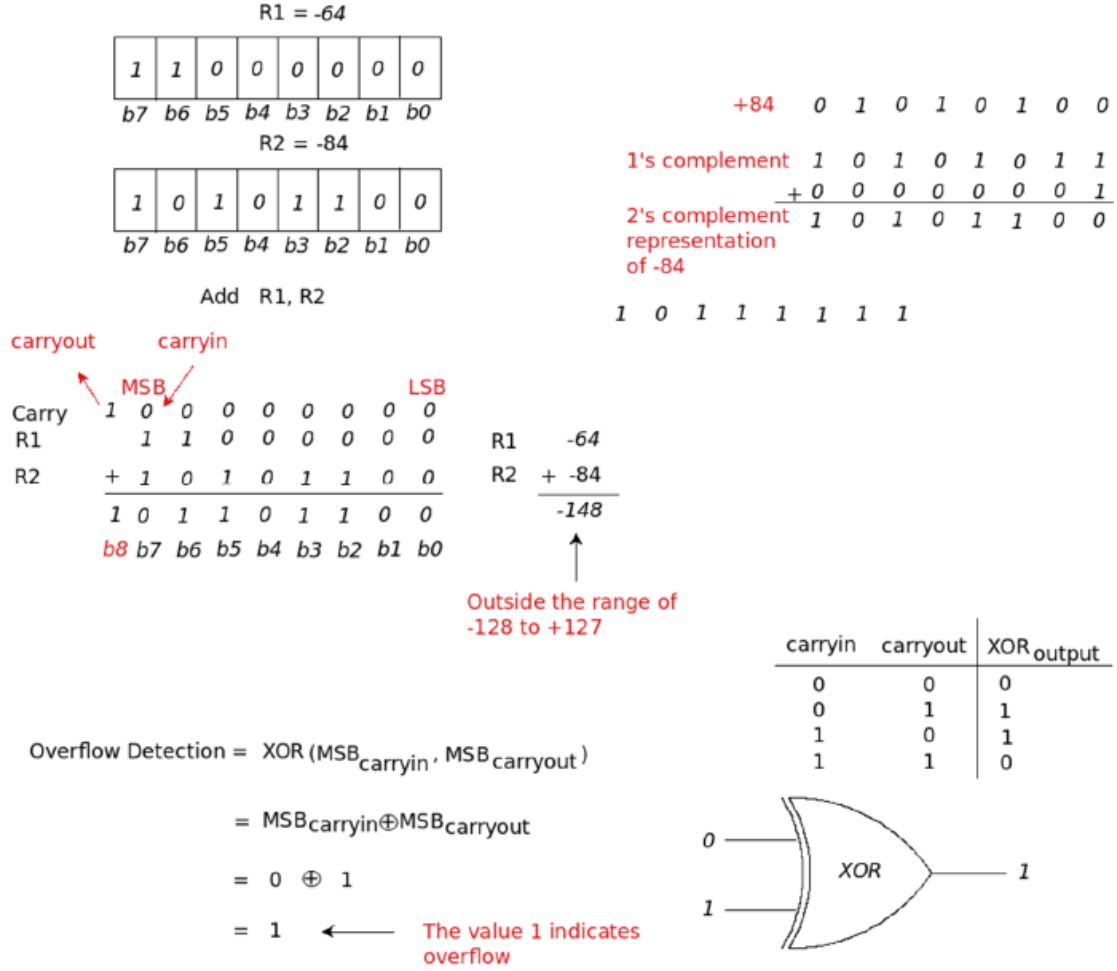
उदाहरण 1.3

2s कॉम्प्लिमेंट संख्या प्रणाली वापरून अनुक्रमे R1 आणि R2 या दोन 8-बिट रजिस्टर मध्ये साठवलेल्या दोन साइन -64 आणि -84 संख्या ची बेरीज करा?

उपाय: प्रत्येक 8-बिट रजिस्टरसाठी स्वीकार्य संख्यांची रेंज -128 ते + 127 असते. -64 आणि -84 जोडण्याचा परिणाम म्हणजे $(-64) + (-84) = -148$. 8-बिट रजिस्टरमध्ये साठवली जाऊ शकणारी जास्तीत जास्त नकारात्मक संख्या -128 आहे, म्हणून हे व्हॅल्यू त्याच्या साठवण मर्यादेपेक्षा जास्त आहे.

-64 आणि -84 ची एरिथमेटिक बेरीज आकृती 1.16. मध्ये दर्शविली आहे. दोन्ही पूर्णांक नकारात्मक आहेत, म्हणून R1 आणि R2 रजिस्टरमध्ये ते 2s कॉम्प्लिमेंट संख्या प्रणालीचा वापर करून दर्शविले जातात. ओव्हरफ्लो ओळखण्यासाठी अंकगणिताच्या जोडणीदरम्यान एमएसबी च्या कॅरी-इन आणि कॅरी-आउट ला एक्सओर(xor) गेटद्वारे दिले जाते. एक्सओर गेटचे आउटपुट 1 असल्या मुळे आपण असें समजू की ओव्हरफ्लो झाला आहे.

परिणामी 8-बिट व्हॅल्यू नकारात्मक असावे. तथापि, b7 बिट सूचित करते की परिणाम एक सकारात्मक संख्या आहे. तथापि, जर साइन बिट स्थितीचा वापर परिणामाचा साइन बिट म्हणून केला गेला तर प्राप्त झालेले 9-बिट उत्तर योग्य असेल. कारण उत्तर 8 बिट्समध्ये हाताळले जाऊ शकत नाही, ओव्हरफ्लो होतो.



आकृती 1.16: दोन ऋण संख्या -64 आणि -84 जोडल्यावर ओव्हरफ्लो डिटेक्शन

1.6.2 फ्लोटिंग पॉइंट क्रमांकाचे प्रतिनिधित्व

इंटेजर च्या उजव्या बाजूला बिट b_0 नंतर बरोबर बायनरी पॉइंट आहे असे गृहीत धरले जाते. 32-बिट वर्ड (word) लांबी असलेल्या संगणकावर 2s कॉम्प्लिमेंट साइन इंटेजर दर्शविण्यासाठी, -2^{31} ते $+2^{31}-1$ ही संख्या वापरली जाते. अपूर्णांक पॉइंट साइन बिटच्या उजव्या बाजूला-बिट्स b_{31} आणि b_{30} मध्ये असावा. अपूर्णांक रेन्ज -1 आणि $+1-2^{-31}$ दरम्यान असते. सगळ्यात कमीत कमी अपूर्णांक ची मर्यादा 10^{-10} आहे.

फिक्स पॉइंट नंबर फॉरमाट वापरलेल्या गणनेवर विज्ञान आणि तंत्रज्ञान क्षेत्रात निर्बंध लादतो. सोयीसाठी बायनरी संख्या मोठ्या पूर्णांक आणि अगदी लहान अपूर्णांक दोन्ही दर्शवू शकतात. संगणकामध्ये संख्यांशी व्यवहार करण्याची आणि त्यांचे प्रतिनिधित्व करण्याची क्षमता अशा प्रकारे असते की बायनरी पॉइंटचे स्थान बदलले जाऊ शकते आणि गणना होत असताना आपोआप अद्ययावत (update) केले जाते. जेव्हा बायनरी पॉइंट त्यांच्या भोवती फिरतो, तेव्हा फ्लोटिंग-पॉइंट पूर्णांकांचा वापर आवश्यक असतो.

फ्लोटिंग-पॉइंट म्हणून बायनरी पॉइंट त्याचे स्थान दर्शवितो. दशांश वैज्ञानिक संकेतना मध्ये पूर्णांक 8.0247×10^{24} , 3.927×10^{-37} , -1.0431×10^4 , -6.8000×10^{-18} , इत्यादी म्हणून व्यक्त केले जातात. त्यांच्याकडे अचूकतेचे पाच महत्त्वपूर्ण अंक आहेत. लक्षणीय अंकांच्या संदर्भात दशांश पॉइंटचे स्थान अनुक्रमे 10^{24} , 10^{-37} , 10^4 आणि 10^{-18} या प्रमाण घटकांचा वापर करून फिक्स्ड केले जाऊ शकते.

चिन्ह (signed), मान्टिसा (mantisaa) आणि घातांक (exponent) हे तीन भाग आहेत जे फ्लोटिंग-पॉइंट नंबर तयार करतात. मान्टिसा हे एकतर अपूर्णांक किंवा पूर्णांक असतात जे फिक्स्ड-पॉइंट स्वरूपाचे असतात. $+51185.987$ ही संख्या एक अपूर्णांक म्हणून लिहिली जाते आणि त्यानंतर फ्लोटिंग-पॉइंट स्वरूपात घातांक लिहिला जातो:

अपूर्णांक	घातांक
$+0.51185987$	$+05$

घातांकाच्या व्हॅलूनुसार, अपूर्णाकाचा वास्तविक दशांश पॉइंट तो जिथे दर्शविला जातो त्याच्या उजवीकडे पाच अंक असतो. हे नोटेशन $+0.51185987 \times 10^{+5}$ या वैज्ञानिक नोटेशनमध्ये वापरले जाते. बेस चे प्रतिनिधित्व करणे अनावश्यक आहे कारण ते फिक्स्ड आहे. सकारात्मक आणि नकारात्मक घटक अस्तित्वात आहेत.

एमएसबी चिन्ह बिटचे प्रतिनिधित्व करते, जेथे 0 ही सकारात्मक संख्या दर्शवते आणि 1 ही नकारात्मक संख्या दर्शवते. एक बेस घातांक घटक आणि एक महत्त्वपूर्ण घटक आहे, जो खालीलप्रमाणे दर्शविला जातो:

$$m \times r^e$$

येथे m मान्टिसा आहे आणि e हा घातांक आहे. रजिस्टरमध्ये त्यांचे बायनरी क्रमांक साठवले जातात, चिन्हां सहित. नेहमी रॅडिक्स आर आणि मान्टिसा रॅडिक्स-पॉइंट स्थिती गृहीत धरा. हे दोन गृहीतक योग्य संगणकीय परिणामांची खात्री करतात. फ्लोटिंग-पॉइंट बायनरी पूर्णाकाचा घातांक बेस 2 आहे. बायनरी संख्या $+1101.01$ मध्ये 8-बिट अपूर्णांक आणि 6-बिट घातांक आहे:

अपूर्णांक	घातांक
01101010	000100

अपूर्णाकाच्या सर्वात डाव्या स्थानावरील 0 हे सकारात्मक चिन्ह सूचित करते. बायनरी क्रमांक +4 हा घातांक व्हॅलू 000100 द्वारे दर्शविला जातो. मंटिसा आणि घातांक दोन्ही असे लिहिले जाऊ शकतात

$$m \times r^e = + (.01101010) \times 2^{+4}$$

मान्टिसा एमएसबी अंक शून्येतर बनवल्याने फ्लोटिंग-पॉइंट व्हॅलू सामान्य होते. 000350 च्या उलट, 350 सामान्यीकृत आहे. मान्टिसाच्या मुळाच्या पॉइंटची पर्वा न करता, जर त्याचा सर्वात डावा अंक शून्येतर असेल तरच ही संख्या सामान्य केली जाते.

00011110 चे सामान्यीकरण करण्यासाठी, ते डावीकडे तीन ठिकाणी हलवा आणि 11110000 मिळविण्यासाठी अग्रगण्य 0 काढून टाका. संख्या 2^3 ने गुणाकार केली जाते, जी आठव्या बरोबरीची आहे. घातांकातून 3 वजा केल्याने फ्लोटिंग-पॉइंट क्रमांक समान राहतो. सामान्यीकृत व्हॅल्यू फ्लोटिंग-पॉइंट अचूकता वाढवतात. शून्यांचे सामान्यीकरण करता येत नाही कारण त्यांना शून्येतर अंक नसतात. मान्दिसा आणि घातांक हे सर्व फ्लोटिंग पॉइंटमध्ये 0 आहेत.

फ्लोटिंग-पॉइंट नंबर्सचा वापर करून केलेल्या एरिथमेटिक ऑपरेशन्ससाठी अधिक क्लिष्ट हार्डवेअरची आवश्यकता असते आणि फिक्स्ड-पॉइंट व्हॅल्यूसह केलेल्या ऑपरेशन्सपेक्षा कार्यान्वित होण्यास जास्त वेळ लागतो. तथापि, फिक्स्ड-पॉइंट गणनेशी संबंधित स्केलिंग समस्यांमुळे, वैज्ञानिक गणनांसाठी फ्लोटिंग-पॉइंट प्रतिनिधित्व आवश्यक आहे. आधुनिक संगणकांमध्ये फ्लोटिंग-पॉइंट एरिथमेटिक गणना करण्याची क्षमता आहे.

32-बिट फ्लोटिंग-पॉइंट पूर्णांक एन्कोडिंगसाठी आयईईई (IEEE) मानक 2 च्या आधारासह स्केलिंग फॅक्टर [8] च्या साइन केलेल्या घातांकासाठी चिन्ह बिट, 23 महत्त्वपूर्ण बिट्स आणि 8 बिट्स वापरते. विज्ञान आणि अभियांत्रिकीमध्ये वापरल्या जाणाऱ्या बहुतांश गणनांसाठी हे पुरेसे आहे. त्याच आयईईई (IEEE) मानकांचे पालन करणारे 64-बिट स्वरूप संभाव्य व्हॅल्यूच्या मोठ्या व्याप्तीव्यतिरिक्त अधिक अचूकता प्रदान करते. या स्वरूपामध्ये अतिरिक्त महत्त्वपूर्ण बिट्स आणि साइन केलेले घातांक बिट्स देखील समाविष्ट आहेत. अध्याय 2 मध्ये फ्लोटिंग-पॉइंट नंबर्सचे प्रतिनिधित्व तसेच फ्लोटिंग-पॉइंट अंकगणिताची चर्चा केली आहे.

1.7 एरर डिटेक्शन कोड

बिट 0 आणि 1 हे दोन भिन्न एनालॉग सिग्नल किंवा व्होल्टेज श्रेणीशी संबंधित आहेत. ध्वनी (noise) हस्तक्षेपामुळे बायनरी डेटा एका प्रणालीतून दुसऱ्या प्रणालीमध्ये प्रसारित करताना हे संकेत बदलू शकतात. ध्वनी (noise) सिग्नल बदलू शकतो आणि इतर प्रणालीद्वारे प्राप्त डेटामध्ये त्रुटी निर्माण करू शकतो. प्राप्तकर्त्याने प्राप्त केलेली माहिती प्रेषकाने पाठवलेल्या माहितीशी जुळत नसल्यास त्रुटी उद्भवते. सिंगल बिटमधील बदल झाल्यास त्याला सिंगल-बिट एरर मानला जातो किंवा एकापेक्षा जास्त बिटमधील बदलाला बिटस्ट्रीम एरर म्हणतात.

उदाहरणार्थ, प्रेषक (sender) $I_s=00101111$ डेटा प्रसारित करतो. ट्रान्समिशन दरम्यान, I_s चा MSB बिट आहे 0 ते 1 बदलले आणि प्राप्तकर्त्याला (receiver) $I_R=10101111$ प्राप्त होतो. ही एकल-बिट त्रुटी (single-bit error) आहे. एक अधिक पेक्षा ज्यास्त बिट बदलला असल्यास, समजा, b_0 , b_1 आणि b_4 पैकी I_s चे बिट बदलले जातात, त्यानंतर रिसीव्हर बिटस्ट्रीम त्रुटीमुळे $I_R=00111100$ प्राप्त करतो.

तक्ता 1.2: सम आणि विषम पॅरिटी कोड प्रतिनिधित्व

बायनरी कोड	सम पॅरिटी बिट	सम पॅरिटी कोड	विषम समता बिट	विषम समता कोड
000	0	0000	1	0001
001	1	0011	0	0010
010	1	0101	0	0100
011	0	0110	1	0111
100	1	1001	0	1000
101	0	1010	1	1001
110	0	1100	1	1101
111	1	1111	0	1110

त्रुटी शोध कोडद्वारे या त्रुटी शोधल्या जाऊ शकतात. पॅरिटी आणि हॅमिंग कोड ही अशी दोन उदाहरणे आहेत. हे कोड मूळ डेटा बिटस्ट्रीमच्या प्रसारणादरम्यान झालेल्या त्रुटी ओळखतात. मूळ बिट प्रवाहात पॅरिटी बिट एकतर सर्वात महत्त्वपूर्ण बिटच्या (एमएसबी) डावीकडे किंवा सर्वात कमी महत्त्वपूर्ण बिटच्या उजवीकडे जोडला जातो. (LSB). हा पॅरिटी बिट अशा प्रकारे निवडला जाऊ शकतो

1. **सम पॅरिटी कोड (Even Parity Code):** जर बायनरी कोडमध्ये 1 सम असल्यास, तर सम पॅरिटी बिट 0 असावा. नसल्यास, ते 1 असावा. येथे सम पॅरिटी कोडमध्ये 4 बिट्स असतात. संभाव्य सम संख्या यामध्ये सम पॅरिटी कोड 0, 2 आणि 4 आहेत.

2. **विषम पॅरिटी कोड (Odd Parity Code):** बायनरी कोडमध्ये 1 विषम असल्यास, विषम पॅरिटी बिट 0 असावा. जर नाही, ते 1 असावे. विषम पॅरिटी कोड 4 बिट्सचे बनलेले असतात. या विषम पॅरिटी कोडमधील विषम संख्या एकतर 1 किंवा 3 असू शकते. जर दुसऱ्या प्रणालीला वापरलेल्या सम किंवा विषम पॅरिटी कोडनुसार पॅरिटी कोडमध्ये निर्दिष्ट संख्या मिळाली, तर त्याचा डेटा योग्य आहे. अन्यथा, डेटा चुकीचा आहे.

दोन्ही कोडिंग योजनांचे प्रतिनिधित्व तक्ता 1.2 मध्ये स्पष्ट केले आहे. व्याख्येनुसार, सम आणि विषम पॅरिटी बिटचे एक-बिट व्हॅल्यू अनुक्रमे कॉलम 2 आणि कॉलम 4 मध्ये गणना केली आहे. बायनरी डेटामध्ये माहितीचे तीन बिट्स असतात, ज्यामध्ये बायनरी कोडच्या सर्वात उजव्या बाजूला सम किंवा विषम पॅरिटीचा एक बिट



स्कॅन करा

सम समता आणि विषम समता
संकेतांच्या यंत्रसामग्री

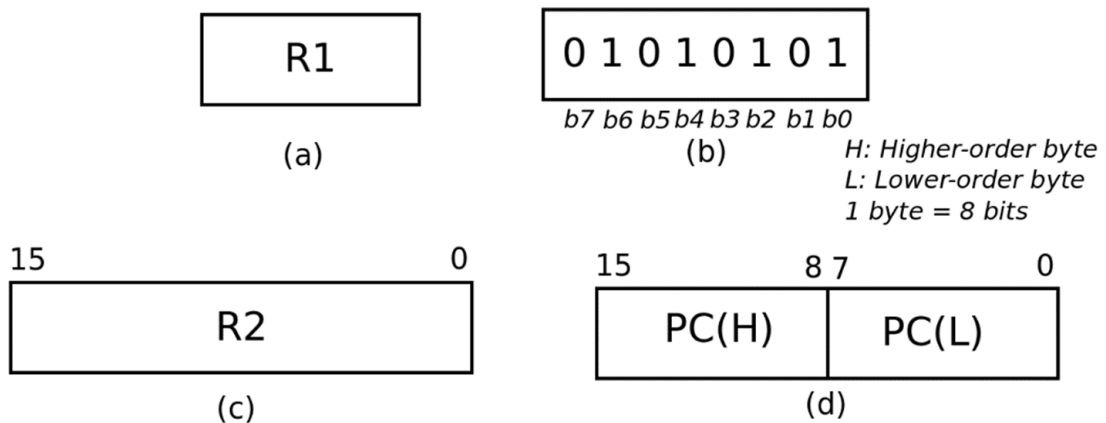
ठेवला जातो. स्तंभ 3 आणि स्तंभ 5 अनुक्रमे सम आणि विषम पॅरिटी कोड सूचीबद्ध करतात. पॅरिटी बिट एकच त्रुटी शोधू शकते परंतु ती दुरुस्त करू शकत नाही. हॅमिंग कोड एक-बिट आणि दोन-बिट त्रुटी शोधू शकतो किंवा एक-बिट त्रुटी दुरुस्त करू शकतो. हे अनेक पॅरिटी बिट्स वापरते.

1.8 रजिस्टर ट्रान्सफर आणि सूक्ष्म ऑपरेशन्स:

मायक्रो-ऑपरेशन ही प्राथमिक ऑपरेशन्स आहेत जी रजिस्टर डेटावर केली जातात. ऑपरेशनचे आउटपुट एकतर एकाच रजिस्टरमध्ये ओव्हरराइट केले जाऊ शकते किंवा वेगळ्या रजिस्टरमध्ये हलवले जाऊ शकते. उदाहरणार्थ ऑपरेशन्स, शिफ्ट, काउंट, क्लिअर आणि लोड ही मायक्रो-ऑपरेशन्स आहेत. निम्न-स्तरीय इंस्ट्रक्शन या सूक्ष्म कार्यपद्धती आहेत ज्या यंत्रातील गुंतागुंतीच्या इंस्ट्रक्शन अंमलात आणण्यासाठी वापरल्या जातात. रजिस्टर ट्रान्सफर लॅंग्वेज (RTL) प्रतीकात्मक स्वरूपात डिजिटल मॉड्यूलच्या इंस्ट्रक्शन मधील सूक्ष्म-ऑपरेशन अनुक्रम दर्शवते. हे डिजिटल संगणकांच्या अंतर्गत संस्थेचे संक्षिप्त वर्णन करते. हे केवळ इंस्ट्रक्शन मधील सूक्ष्म-कार्यांच्या परिणामांची हालचाल व्यक्त करत नाही, तर इंस्ट्रक्शन आणि मेमोरी मधील डेटाचे ट्रान्सफर देखील दर्शवते.

1.8.1 रजिस्टर ट्रान्सफर

हार्डवेअर लॉजिक सर्किट्सद्वारे रजिस्टर ट्रान्सफर सुलभ केले जाते जे विशिष्ट सूक्ष्म कार्ये (micro-operation) करतात आणि परिणाम समान किंवा वेगळ्या रजिस्टर मध्ये ट्रान्सफर करतात. रजिस्टर ट्रान्सफर रिप्लेसमेंट ऑपरेटर (\leftarrow) द्वारे प्रतीकात्मकपणे प्रस्तुत केले जाते. उदाहरणार्थ, $R2 \leftarrow R1$ हे विधान, $R2$ रजिस्टर करण्यासाठी रजिस्टर $R1$ पासून सामग्रीचे ट्रान्सफर परिभाषित करते.



आकृती. 1.17: रजिस्टर्समधील व्हॅल्यूचे भिन्न प्रतिनिधित्व

रजिस्टर ट्रान्सफर वेगवेगळ्या प्रकारे परिभाषित केले जाऊ शकते:

तक्ता 1.3: रजिस्टर ट्रान्सफरचे सिम्बॉलिक रेप्रेसेंटेशन

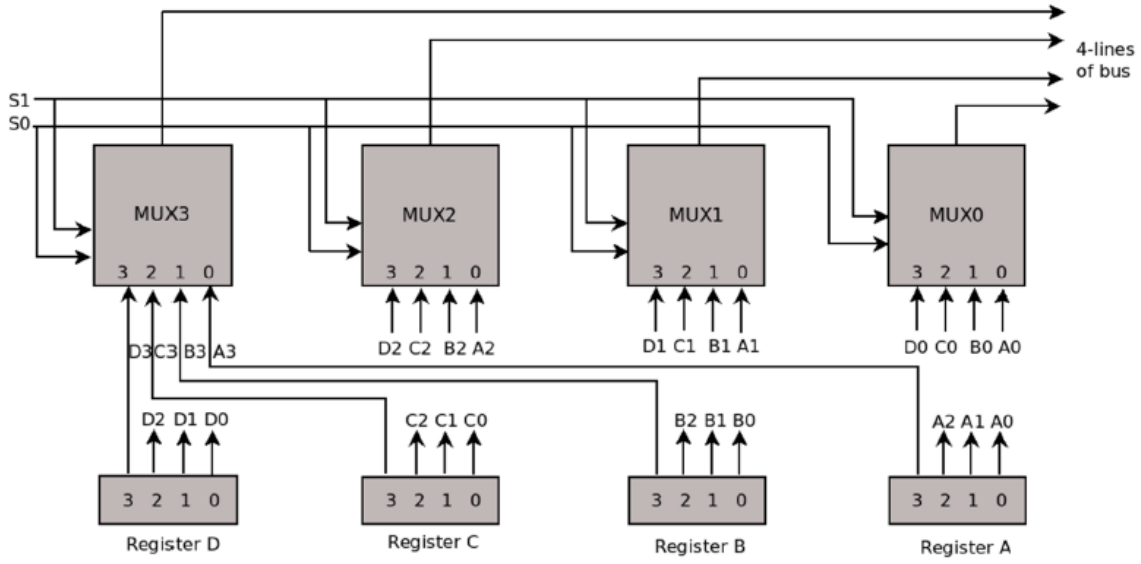
चिन्ह	वर्णन	उदाहरण
अक्षरे आणि संख्या	रजिस्टर दर्शवते	R1, R2, MAR
() किंवा []	रजिस्टरचा एक भाग दर्शवा	R ₂ (8) किंवा R ₂ [8] , R ₂ (0-7) किंवा R ₂ [0-7]
←	माहितीचे ट्रान्सफर दर्शवा	R2 ← R1
:	कंडिशनल ऑपरेशन दर्शवा	C: R2 ← R1 If C=1

- तक्ता 1.3 मध्ये दर्शविल्याप्रमाणे, सामान्यतः आयताकृती बॉक्स किंवा कंसात जोडलेल्या रजिस्टर च्या नावाने रजिस्टर दर्शविले जातात.
- याव्यतिरिक्त, विशिष्ट बिट्स कंसात जोडून हायलाइट केले जाऊ शकतात.
उदाहरणार्थ, आकृती 1.17 (b) मध्ये. R2 (8) बिट b₈ दर्शवितो, किंवा PC (8-15) बिटस्ट्रीम b₈ ते b₁₅ आकृती 1.17 (c) आणि आकृती. 1.17(d). मध्ये दर्शवितो.
- आकृती 1.17(a) मध्ये दर्शविल्याप्रमाणे रजिस्टर R₀ ते R (n-1) किंवा R₀ ते R (n-1) पर्यंत क्रमांकित आहेत.
- आकृती 1.17(c) मध्ये आणि आकृती. 1.17(d) मध्ये दर्शविल्याप्रमाणे, रजिस्टरमधील बिट क्रमांकन चौकटीच्या वरच्या स्थानी दर्शविले आहे.
- 16-बिट रजिस्टर पीसीच्या बिट्सना (0 ते 7) 16-बिट ऍड्रेसच्या खालच्या बाइटस नियुक्त केले जातात, तर बिट्सना (8 ते 15) आकृतीमध्ये 1.17(d). दर्शविल्याप्रमाणे 16-बिट ऍड्रेसच्या उच्च बाइटस नियुक्त केले जातात.



स्कॅन करा
बस आणि मेमोरी ट्रान्सफर
हार्डवेअर सर्किटसाठी

तक्ता 1.3 मध्ये रजिस्टर ट्रान्सफरच्या वेळी वापरल्या जाणाऱ्या रजिस्टर ची वेगवेगळी प्रतिकात्मक सादरीकरणे दर्शविली आहेत. याव्यतिरिक्त, कंडिशनल रजिस्टर ट्रान्सफरचे प्रतिनिधित्व देखील केले जातात. उदाहरणार्थ, C: R2 ← R1 ही अभिव्यक्ती एका विशिष्ट कंट्रोल कार्याखाली R1 रजिस्टर वरून R2 रजिस्टर करण्यासाठी डेटा ट्रान्सफर परिभाषित करते. (C). जर (C = 1) तर रजिस्टर R1 चा डेटा रजिस्टर R2 (R2 ← R1) मध्ये ट्रान्सफर केला जाईल. कारण C = 1 सूचित करते की कंट्रोल युनिट या रजिस्टरमध्ये असे डेटा ट्रान्सफर सक्रिय करण्यासाठी कंट्रोल सिग्नल तयार करते. रजिस्टर ट्रान्सफर ऑपरेशनपासून कंट्रोल व्हेरिएबल्स वेगळे करून, कंट्रोल फंक्शन निर्दिष्ट करणे सोपे आहे. (C).



आकृती. 1.18: रजिस्टरमधून बसद्वारे माहितीचे ट्रान्सफर

1.8.2 बस आणि मेमोरी ट्रान्सफर

डिजिटल संगणकामध्ये, प्रत्येक बिट किंवा व्हॅल्यू रजिस्टरमध्ये साठवले जाते. जर प्रत्येक रजिस्टरला प्रत्येक रजिस्टरशी जोडण्यासाठी स्वतंत्र तारांचा वापर केला गेल्या, तर परिणामी वायरची जटिलता गंभीर असेल. रजिस्टरमधील माहिती सामायिक (common) बसद्वारे पाठवली जाते. बायनरी डेटा एका रजिस्टरमधून दुसऱ्या रजिस्टरमध्ये नेण्यासाठी, बसची रचना प्रत्येक बिटसाठी एक अशा रेषा किंवा तारांची मालिका वापरते. प्रत्येक रजिस्टर ट्रान्सफरदरम्यान, बस कंट्रोल सिग्नल च्या आधारे डेस्टिनेशन (destination) रजिस्टर निवडले जातात.

मल्टीप्लेक्सर्सद्वारे, ज्या रजिस्टरची माहिती सामान्य बसमध्ये प्रसारित केली जाऊ शकते ती बसशी जोडली जातात. प्रत्येक रजिस्टरमध्ये n बिट्स असतात, ज्यांना 0 ते $n-1$ क्रमांकित केले जाते. ही बस n मल्टीप्लेक्सरने बनलेली आहे, प्रत्येक मल्टीप्लेक्सरमध्ये डेटा इनपुट (0 ते $n-1$) आणि $n = 2^m$ असते, परिणामी m निवड रेषा (selection line) असतात.

प्रत्येक रजिस्टरमधील बिट्स, ज्यांचे सर्व समान महत्त्वपूर्ण स्थान आहे, एक बस मार्ग तयार करण्यासाठी एका मल्टीप्लेक्सरच्या डेटा इनपुटशी जोडले जातात. त्याचप्रमाणे, प्रत्येक रजिस्टरचे बिट्स b_0 ते b_{n-1} योग्य मल्टीप्लेक्सरशी जोडले जातात. सर्व मल्टीप्लेक्सर्सचे निवड इनपुट (selection line) m निवड रेषांशी (selection line) जोडलेले असतात.

उदाहरणार्थ, जेव्हा $m = 4$ आणि $n = 2$, तेव्हा रजिस्टरमधून चार बिट्स निवडले जातात आणि चार ओळींच्या सामायिक बसमध्ये प्रसारित केले जातात. कारण $n = 4$, एकूण चार मल्टीप्लेक्सर्स (MUX) आवश्यक आहेत. चार स्वतंत्र



स्कॅन करा

प्रोसेसर ॲड्रेसिंग मोड
जाणून घेण्यासाठी

रजिस्टर, i.e., रजिस्टर A, B, C आणि D या प्रकरणात चार मल्टीप्लेक्सर्सद्वारे बसशी जोडलेले आहेत. जेव्हा $S_1S_0 = 00$, तेव्हा बस आउटपुट सर्व मल्टीप्लेक्सर्सच्या प्रारंभिक डेटा इनपुटमधून काढले जातात, अर्थात रजिस्टर A साठी, बिट्स A_0, A_1, A_2 आणि A_3 जे MUX0, MUX1, MUX2 आणि MUX3 चे पहिले इनपुट आहेत. प्रत्येक एमयूएक्स (MUX) चे आउटपुट एक बस मार्ग आहे. परिणामी, सर्व चार बिट्स बसद्वारे टारगेट रजिस्टर मध्ये प्रसारित केले जातात. जर S_1S_0 हा 01 असेल, तर दुसऱ्या रजिस्टरमधील (रजिस्टर B) डेटा बसमध्ये पाठविला जातो. त्याचप्रमाणे, आकृती 1.18 मध्ये दर्शविल्याप्रमाणे अनुक्रमे $S_1S_0 = 10$ आणि 11 असताना C आणि D रजिस्टरमधून डेटा पाठविला जातो.

मेमोरी ट्रान्सफर *Memory Transfer*

मेमोरी ट्रान्सफर हे इतर घटकांद्वारे पटवून दिलेल्या रीड आणि राइट क्रियांचे वर्णन करते. जेव्हा माहिती मेमोरी युनिटमधून अंतिम वापरकर्त्याकडे हलवली जाते तेव्हा त्या प्रणालीला रीड ऑपरेशन म्हणून संबोधले जातात. राइट ऑपरेशन म्हणजे मेमोरी मध्ये नवीन माहिती संचयित करण्याची प्रक्रिया आहेत. रजिस्टरचे नाव किंवा जेथे ती साठवली जाते त्या मेमोरी चा ऍड्रेस देऊन माहिती दोन प्रकारे मिळवता येतात. लोड आणि स्टोअर इंस्ट्रक्शन मेमोरी पर्यंतचे रीड आणि राइट कार्य अनुक्रमे हाताळतात.

इंस्ट्रक्शन रीड ऑपरेशन दरम्यान प्रथम लक्ष्य मेमोरी ऍड्रेस, समजा एडीआर, मेमोरीला प्रदान करून आणि नंतर संबंधित कंट्रोल सिग्नल सक्रिय करून मेमोरीपासून प्रोसेसरपर्यंत डेटा ट्रान्समिशन सुरू करतात. त्यानंतरच्या टप्प्यात, माहिती मेमोरीतून रीड केली जाते. त्याचप्रमाणे, मेमोरी मध्ये लिहिताना, वर्डचा ऍड्रेस मेमोरीशी संप्रेषित (communicate) करणे आवश्यक आहे. उदाहरणार्थ, सध्याच्या ADD इंस्ट्रक्शन मेमोरीतून रीड केली जाते.

Add R3, R1, R2

त्यानंतर काय ऑपरेशन करायचे आहे हे ठरवण्यासाठी कंट्रोल युनिट ते डीकोड करते. डीकोडिंग दरम्यान, असे आढळून आले आहे की मेमोरी लोकेशनस ADR आणि ADR1 मधील डेटा अनुक्रमे R1 आणि R2 रजिस्टरमध्ये लोड करणे आवश्यक आहे.

Load R1, ADR

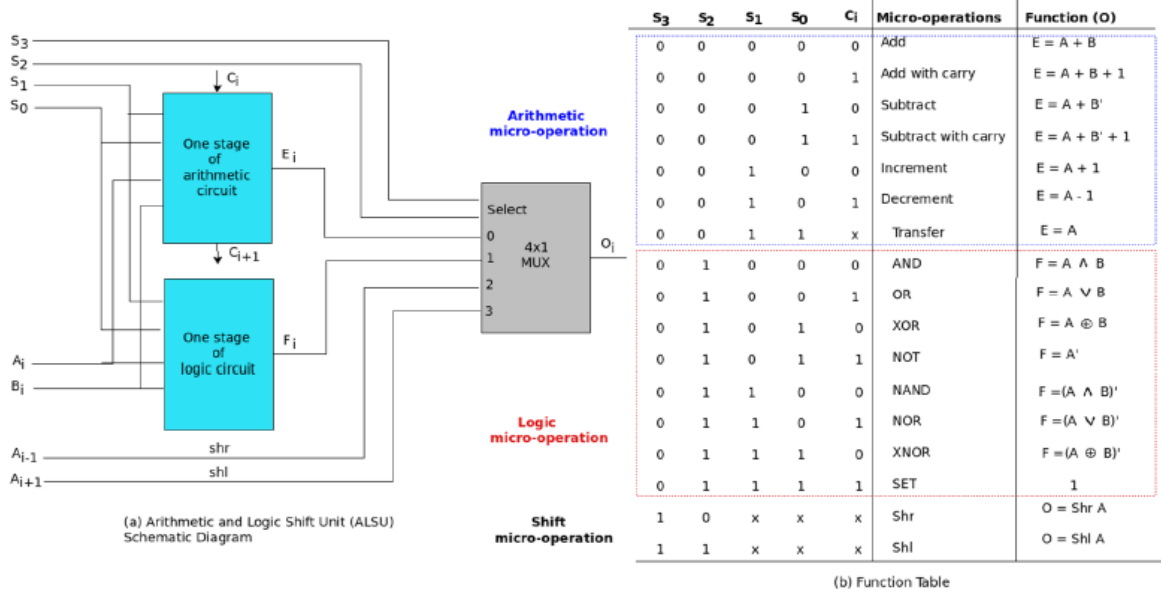
Load R2, ADR1

लोड इंस्ट्रक्शन वापरून, मेमोरी लोकेशनस ADR आणि ADR1 ची सामग्री अनुक्रमे प्रोसेसर रजिस्टर R1 आणि R2 मध्ये रीड आणि लोड केली जाते. ऑपरेंड लोड केल्यानंतर, कंट्रोल युनिट्स एएलयू ला अरीथमेटिक ADD ऑपरेशन सिग्नल पाठवतात.

Add इंस्ट्रक्शन रजिस्टर R1 आणि रजिस्टर R2 च्या व्हॅल्यूची अडिशन करते आणि निकाल रजिस्टर R3 मध्ये सेव्ह करते. स्टोर इंस्ट्रक्शन चा वापर करून, रजिस्टर R3 ची व्हॅल्यू मेमोरीवर लिहिले जाऊ शकते.

Store ADR, R3

स्टोर इंस्ट्रक्शन कार्यान्वित करून, रजिस्टर R3 मधील डेटा मेमोरी ADR मधील ऍड्रेस वर लिहिला जातो. रजिस्टर R3 ची व्हॅल्यू ADR स्थानावरील मजकुरावर लिहिले जाते.



आकृती. 1.19 अरीथमेटिक आणि लॉजिक शिफ्ट युनिट मायक्रोऑपरेशन्स योजनाबद्ध आकृती आणि कार्य सारणी

1.8.3 अरीथमेटिक आणि शिफ्ट लॉजिक युनिट

डिजिटल संगणकाच्या सूक्ष्म क्रियांमध्ये अरीथमेटिक, लॉजिक आणि शिफ्ट ऑपरेशन्सचा समावेश होतो. ही सर्व सूक्ष्म कार्ये डिजिटल सर्किट्सवर केली जातात. अरीथमेटिक लॉजिक शिफ्ट युनिट (एएलएसयू) सर्व सूक्ष्म क्रियांना एकाच सर्किट्स मध्ये समाकलित करते. संगणक प्रणालींमधील अनेक स्टोरेज रजिस्टर एएलएसयू शी जोडलेले आहेत.

एएलएसयू एकाच क्लॉक पल्स मध्ये कार्य करू शकते, नंतर निकाल डेस्टिनेशन रजिस्टरमध्ये ट्रान्स्फर करू शकतात. जेव्हा शिफ्ट मायक्रो-ऑपरेशन्स एएलएसयू मध्ये एकत्रित केले जातात तेव्हा अरीथमेटिक लॉजिक शिफ्ट युनिट तयार केले जाते. त्यामुळे एएलएसयू हा एएलएसयू चा एक भाग आहे.

एएलएसयू चे प्राथमिक कार्य म्हणजे सर्व लॉजिक, अरीथमेटिक आणि शिफ्ट ऑपरेशन्स करणे जे सामायिक निवड चलांसह एकाच एएलएसयू मध्ये एकत्र करतात. अरीथमेटिक एकक बेरीज, वजाबाकी, गुणाकार आणि भागाकार

करतात. लॉजिकल ऑपरेशन म्हणजे संख्या आणि विशेष वर्णावरील ऑपरेशन्स आणि दोन किंवा अधिक माहिती वाक्ये जोडणे. शिफ्ट मायक्रोऑपरेशन्स हे अनुक्रमिक माहिती ट्रान्स्फर ऑपरेशन्स आहेत.

आकृती 1.19 (अ) मध्ये ALSU च्या एकाच टप्प्याची आकृती दर्शविली आहे. सबस्क्रिप्ट i एक मानक टप्पा दर्शविते. n -बिट संख्येसाठी i ची श्रेणी 0 ते $n-1$ टप्पे आहे. $S1$ आणि $S0$ इनपुटसह, एक विशिष्ट मायक्रोऑपरेशन निवडले जातात. 4×1 मल्टीप्लेक्सर अरीथमेटिक आउटपुट, लॉजिक आउटपुट, शिफ्ट राईट आणि शिफ्ट लेफ्ट आउटपुटमध्ये निवडतो. मल्टीप्लेक्सरमध्ये इनपुट $S3$ आणि $S2$ डेटा निवडतात. मल्टीप्लेक्सरच्या इतर दोन डेटा इनपुट शिफ्ट-राईट ऑपरेशनसाठी (shr) $Ai-1$ आणि शिफ्ट-लेफ्ट ऑपरेशनसाठी $Ai + 1$ प्राप्त करतात. (shl).

N -बिट एएलयू ने सर्किटची पुनरावृत्ती करणे आवश्यक आहे. एका अरीथमेटिकच्या स्टेप चे Ci मधील कॅरी त्याच्या $Ci + 1$ च्या कॅरीआउटशी जोडलेले असणे आवश्यक आहेत.

आधीच्या आकृतीमध्ये दर्शविलेले एक-टप्प्याचे सर्किट सात एरिथमेटिक, आठ लॉजिक आणि दोन शिफ्ट ऑपरेशन्स करतात. प्रत्येक ऑपरेशन निवडण्यासाठी $S3$, $S2$, $S1$, $S0$ आणि Ci हे वापरले जातात. येथे, Ci चा वापर केवळ अरीथमेटिक कार्यासाठी केला जातो.

आकृती 1.19 (b) मधील तक्ता अरीथमेटिक लॉजिक शिफ्ट युनिटचे कार्य तक्ता दर्शविते. सात एएलयू ऑपरेशन्स, आठ लॉजिकल ऑपरेशन्स आणि दोन शिफ्ट ऑपरेशन्स आहेत. सुरुवातीचे सात अरीथमेटिक कार्ये आहेत ($S3S2 = 00$). पुढील $S3S2 = 01$ हे आठ लॉजिक कार्ये निवडण्यासाठी वापरले जाते. $S3S2 = 10$ आणि $S3S2 = 11$ हे शिफ्ट ऑपरेशन्स अंतिम दोन ऑपरेशन्स म्हणून निवडण्यासाठी वापरले जातात. उर्वरित तीन इनपुटचा शिफ्टवर कोणताही परिणाम होत नाही. आगामी उपविभागांमध्ये एरिथमेटिक, लॉजिक आणि शिफ्ट मायक्रो-ऑपरेशन्सवर अधिक चर्चा केली आहे.

अरीथमेटिक सूक्ष्म कार्ये (Arithmetic Micro-operations)

अरीथमेटिक सूक्ष्म क्रियांचे तक्ता 1.4 मध्ये सूचीबद्ध केल्याप्रमाणे विविध श्रेणींमध्ये वर्गीकरण केले आहे. बेरीज, वजाबाकी, $1s$ कॉम्प्लिमेन्ट, $2s$ कॉम्प्लिमेन्ट, इनक्रेमेन्ट आणि डिक्रेमेन्ट ही मूलभूत अरीथमेटिक च्या सूक्ष्म क्रिया आहेत. याव्यतिरिक्त, रजिस्टर $R1$ आणि $R2$ मध्ये संग्रहित दोन संख्या एएलयू वापरून जोडल्या जातात आणि परिणाम कोणत्याही रजिस्टरमध्ये संग्रहित केला जातो, समजा रजिस्टर $R3$ मध्ये. वजा ऑपरेशनसाठी, दुसऱ्या क्रमांकाचे $2s$ कॉम्प्लिमेन्ट पहिल्या संख्येमध्ये जोडले जाते. त्यामुळे एएलयू हे दोन्ही ऑपरेशन कॉम्बिनेशन सर्किटद्वारे सहजपणे करू शकते.



स्कॅन करा

अरीथमेटिक सूक्ष्मसक्रियता
समजून घेण्यासाठी

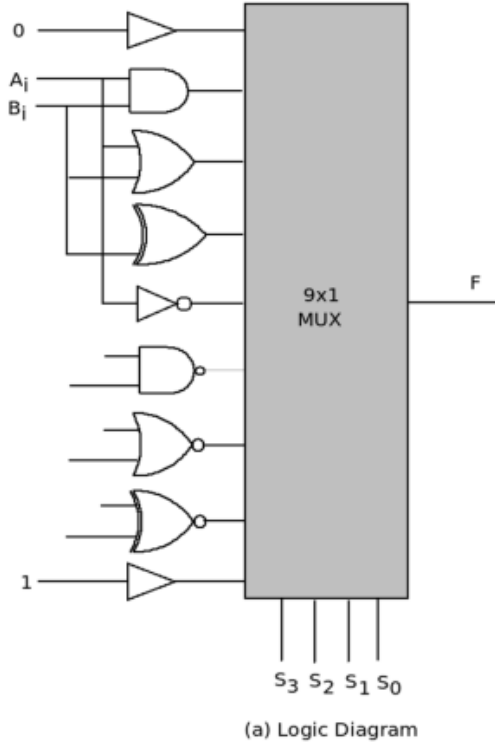
बायनरी अप-डाउन काउंटरचा वापर मायक्रो-ऑपरेशन्स अंमलात आणण्यासाठी केला जातो जे अनुक्रमे अधिक एक (वाढ) आणि वजा एक (घट) ऑपरेशन्स करतात. बहुतेक संगणक ADD आणि SHIFT मायक्रोऑपरेशन्सच्या मालिकेद्वारे गुणाकार करतात. विभाजित करण्यासाठी वजा आणि शिफ्ट सूक्ष्म क्रियांची मालिका वापरली जाते. या परिस्थितीत हार्डवेअर निर्दिष्ट करण्यासाठी जोड, वजा आणि शिफ्टच्या सूक्ष्म-कार्यांची मालिका आवश्यक आहे.

तक्ता 1.4: अरीथमेटिक सूक्ष्म क्रिया

अरीथमेटिक क्रिया	प्रतीकात्मक प्रतिनिधित्व	वर्णन
बेरीज	$R3 \leftarrow R1 + R2$	R1 ची कन्टेन्ट अधिक आर 2 ची कन्टेन्ट रजिस्टरमध्ये आर 3 मध्ये संग्रहित आहे
वजाबाकी	$R3 \leftarrow R1 - R2$	R1 ची कन्टेन्ट वजा R2 ची कन्टेन्ट रजिस्टरमध्ये R3 मध्ये ट्रान्स्फर केली जाते
1 कॉम्पलमेंट	$R2 \leftarrow R2'$	R2 रजिस्टर च्या कन्टेन्ट चे कॉम्पलमेंट करणे
2 कॉम्पलमेंट	$R2 \leftarrow R2' + 1$	R2 रजिस्टर च्या कन्टेन्ट चे 2 कॉम्पलमेंट करणे
वजाबाकी	$R2 \leftarrow R1 + R2' + 1$	R1 ची कन्टेन्ट आणि R2 रजिस्टर च्या कन्टेन्ट चे 2 कॉम्पलमेंट बेरीज
इन्क्रेमेण्ट	$R1 = R1 + 1$	R1 ची कन्टेन्ट 1 ने वाढविणे
डिन्क्रेमेण्ट	$R1 = R1 - 1$	R1 ची कन्टेन्ट 1 ने कमी करणे

लॉजिक मायक्रो ऑपरेशन्स

लॉजिक मायक्रोऑपरेशन्स वैयक्तिक बिट्स हाताळू शकतात. हे ऑपरेशन बिट व्हॅल्यू बदलू, हटवू आणि समाविष्ट करू शकतात. आकृती 1.20 (अ) मध्ये 9×1 मल्टीप्लेक्सरशी जोडलेले नऊ लॉजिक गेट्स स्पष्ट करते. गेटचे आउटपुट हे मल्टीप्लेक्सरचे इनपुट बनते. S3, S2, S1 आणि S0 च्या इनपुट व्हॅल्यूनुसार, लॉजिक गेटचे आउटपुट मल्टीप्लेक्सर आउटपुटमध्ये प्रसारित केले जाते. आकृती 1.20 मध्ये सबस्क्रिप्ट i सह एक सामान्य टप्पा दर्शवते. n बिट्स इनपुटच्या प्रत्येक बिटसाठी, ते n वेळा पुनरावृत्ती केले जातात.



S_3	S_2	S_1	S_0	Logic micro-operation	Function (F)
0	0	0	0	CLEAR	0
0	0	0	1	AND	$A \wedge B$
0	0	1	0	OR	$A \vee B$
0	0	1	1	XOR	$A \oplus B$
0	1	0	0	NOT	A'
0	1	0	1	NAND	$(A \wedge B)'$
0	1	1	0	NOR	$(A \vee B)'$
0	1	1	1	XNOR	$(A \oplus B)'$
1	0	0	0	SET	1

(b) Function Table

आकृती 1.20 लॉजिक मायक्रो ऑपरेशन आणि फंक्शन टेबलची योजनाबद्ध आकृती

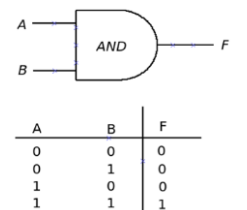
सेलेकटेड लाईन सर्व टप्प्यांशी जोडलेल्या असतात. आकृतीतील तक्ता. 1.20 (b) मध्ये लॉजिक मायक्रोऑपरेशन्स दर्शविते जी रजिस्टरच्या बायनरी डेटावर केली जाऊ शकतात. प्रत्येक बिटवर स्वतंत्रपणे उपचार केले जातात. येथे, A आणि B हे रजिस्टर आहेत ज्यामध्ये डेटा संग्रहित केला जातो आणि F निवडलेले लॉजिक मायक्रो-ऑपरेशन्स केल्यानंतर आउटपुट सेव्ह करतात. प्रत्येक सूक्ष्म-कृतीची चर्चा खाली त्यांच्या सत्य सारण्यांसह पुढील क्रमाने केली आहे:

1. क्लियर (Clear)

रजिस्टर क्लियर करण्यासाठी किंवा रजिस्टरचे बिट्स 0 वर सेट करण्यासाठी क्लियर लॉजिक मायक्रो-ऑपरेशन वापरले जाते.

2. अँड (AND)

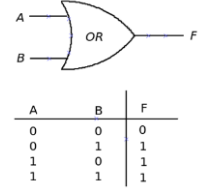
AND लॉजिक मायक्रो-ऑपरेशन लॉजिकदृष्ट्या दोन रजिस्टर्समध्ये असलेल्या डेटाचे बिट्स ANDs करते. लॉजिकल AND \wedge हे चिन्ह द्वारे दर्शविले जाते. जर दोन्ही रजिस्टर A आणि B सत्य असतील, तर AND ऑपरेशनचा परिणाम 1 आहे; अन्यथा, तो 0 आहे.



$F \leftarrow A \wedge B$ हे निर्दिष्ट करते की रजिस्टर A आणि B ला AND सूक्ष्म-ऑपरेशनला संदर्भित केले जाईल, ज्याचा परिणाम रजिस्टर्स F मध्ये ठेवला जाईल. AND लॉजिक मायक्रो-ऑपरेशनचा परिणाम A AND B रजिस्टरच्या इनपुट व्हॅल्यूवर आधारित ट्रुथ टेबल मध्ये दर्शविला आहे.

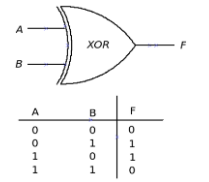
3. ऑर (OR)

OR लॉजिक मायक्रो-ऑपरेशन दोन रजिस्टरच्या बिट्स दरम्यान लॉजिकल OR करते. लॉजिकल OR चे \vee चिन्ह आहे. जर A रजिस्टरचे व्यालू खरे (true) असेल आणि B रजिस्टर चुकी (false) चे असेल, किंवा A रजिस्टरचे व्हॅलू खोटे (false) असेल आणि B रजिस्टरचे व्हॅलू खरे (true) असेल, किंवा A आणि B रजिस्टरचे दोन्ही व्हॅलू खरे (true) असतील, तर OR ऑपरेशनचा परिणाम 1 असेल, अन्यथा तो 0 असेल. ऑपरेशन $F \leftarrow A \vee B$ हे निर्दिष्ट करते की रजिस्टर A आणि B मधील व्हॅलू OR मायक्रो-ऑपरेशनच्या अधीन असतील, आउटपुट रजिस्टर F मध्ये संग्रहित असेल. ट्रुथ टेबल रजिस्टर ए आणि बी इनपुट व्हॅल्यूवर आधारित आउटपुट दर्शवते.



4. एक्सक्लुसिव्ह ऑर (Exclusive OR)

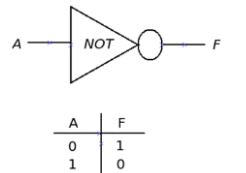
हे लॉजिक मायक्रो-ऑपरेशन, ज्याला एक्सऑर असेही म्हणतात, दोन रजिस्टरच्या बिट्स दरम्यान लॉजिकल एक्सऑर आयोजित करते. लॉजिकल एक्सऑर सूचित करते की एकतर A किंवा B सत्य (truth) असले पाहिजे, परंतु दोन्ही नाही. एक्सक्लुसिव्ह OR हे \oplus चिन्हाद्वारे दर्शविले जाते. A आणि B रजिस्टरमधील व्हॅलू एक्सऑर मायक्रो-ऑपरेशनच्या अधीन असतील, आउटपुट रजिस्टर एफ मध्ये संग्रहित असेल. ट्रुथ टेबल दर्शवते की जेव्हा $A = 1$ आणि $B = 0$, किंवा जेव्हा $A = 0$ आणि $B = 1$, तेव्हा आउटपुट 1 असते.



5. कॉम्प्लिमेंट किंवा नॉट (Complement or NOT)

कॉम्प्लिमेंट लॉजिक मायक्रो-ऑपरेशन इनपुट रजिस्टर A ची कॉम्प्लिमेंट सामग्री आउटपुट रजिस्टर F मध्ये ट्रान्सफर करते. प्रथम, रजिस्टरमधील व्हॅलू कॉम्प्लिमेंट केली जाते आणि नंतर इच्छित रजिस्टरमध्ये हलवली जाते.

ट्रुथ टेबल, $F \leftarrow A'$ हे रजिस्टर A चे कॉम्प्लिमेंट व्हॅलू दर्शवते, जे रजिस्टर F मध्ये हलविले जाते. तर ट्रुथ टेबल A रजिस्टरच्या घेतलेल्या व्हॅलूच्या अगदी उलट आहे.

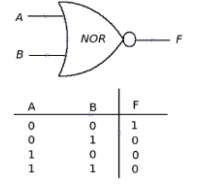


6. नांड (NAND)

NAND लॉजिक मायक्रोऑपरेशन हे AND लॉजिक मायक्रोऑपरेशनच्या उलट आहे. नावाप्रमाणेच, ते 'नॉट अँड' आहे. AND च्या उलट, NAND मध्ये, आउटपुट 0 असते जेव्हा A रजिस्टर आणि B रजिस्टर दोन्हीची व्हॅल्यू 1 असते आणि जेव्हा एकतर A फॉल्स असते किंवा B फॉल्स असते किंवा दोन्ही फॉल्स असतात तेव्हा ते 1 असते, ट्रुथ टेबल दर्शविल्याप्रमाणे $F \leftarrow (A \wedge B)'$.

7. नॉर NOR

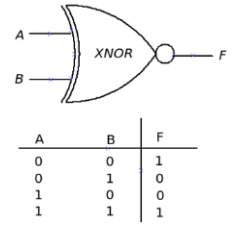
नॉर लॉजिक मायक्रो-ऑपरेशन नॉटऑर. (NOT OR) आहे. OR च्या उलट, NOR मध्ये, जेव्हा A रजिस्टर किंवा B रजिस्टर किंवा दोन्ही A आणि B रजिस्टरचे व्हॅल्यू खरे असते तेव्हा आउटपुट 0 असते आणि जेव्हा A आणि B दोन्ही रजिस्टर चुकीचे असतात तेव्हा ते 1 असते.



$F \leftarrow (A \vee B)'$ ही अभिव्यक्ती इनपुट A आणि B रजिस्टरच्या वेगवेगळ्या व्हॅल्यूसाठी नॉर NOR लॉजिक मायक्रोऑपरेशनची दृढ टेबल दर्शवते.

8. एक्सकलुसिव्ह नॉर Exclusive NOR

जेव्हा A आणि B या दोन्ही रजिस्टरची व्हॅल्यू समान असतात तेव्हा एक्सक्लुसिव्ह NOR (XNOR) मायक्रो-ऑपरेशन आउटपुट 1 सेट करते. ते खरे (true) किंवा खोटे (false) असू शकतात, परंतु ते सारखेच असले पाहिजेत. एक्सनॉर लॉजिक मायक्रो-ऑपरेशन $F \leftarrow (A \oplus B)'$ 'चे दृढ टेबल दर्शवितो की जेव्हा एकतर $A = 0$ आणि $B = 0$ किंवा $A = 1$ आणि $B = 1$ असेल तेव्हा आउटपुट 1 असेल.

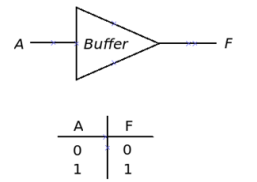


9. Set to all 1's (सर्व 1 वर सेट करा)

सर्व लॉजिक मायक्रो-ऑपरेशन्सचा सेट सर्व रजिस्टर बिट्स 1 वर सेट करण्यासाठी वापरला जातो. उदाहरणार्थ, $F \leftarrow 1$ म्हणजे रजिस्टर F ची व्हॅल्यू 1 वर सेट केलेली आहे. रजिस्टर F चे पूर्वीचे व्हॅल्यू (VALUE) काढून टाकले जाईल.

10. Transfer A

ट्रान्सफर A लॉजिक मायक्रो-ऑपरेशन हे रजिस्टर A चा डेटा आउटपुट रजिस्टर F मध्ये ट्रान्सफर करते. 'ट्रान्सफर A' लॉजिक मायक्रो-ऑपरेशनचे दृढ टेबल दर्शवितो की या मायक्रोऑपरेशनमध्ये रजिस्टर A पासून आउटपुट रजिस्टरमध्ये डेटाचे ट्रान्सफर होते, त्याचे दृढ टेबल रजिस्टर A च्या घेतलेल्या व्हॅल्यूसारखेच आहे. जर A हा 0 असेल तर F हा 0 असेल, जर A हा 1 असेल तर F हा 1 असेल.



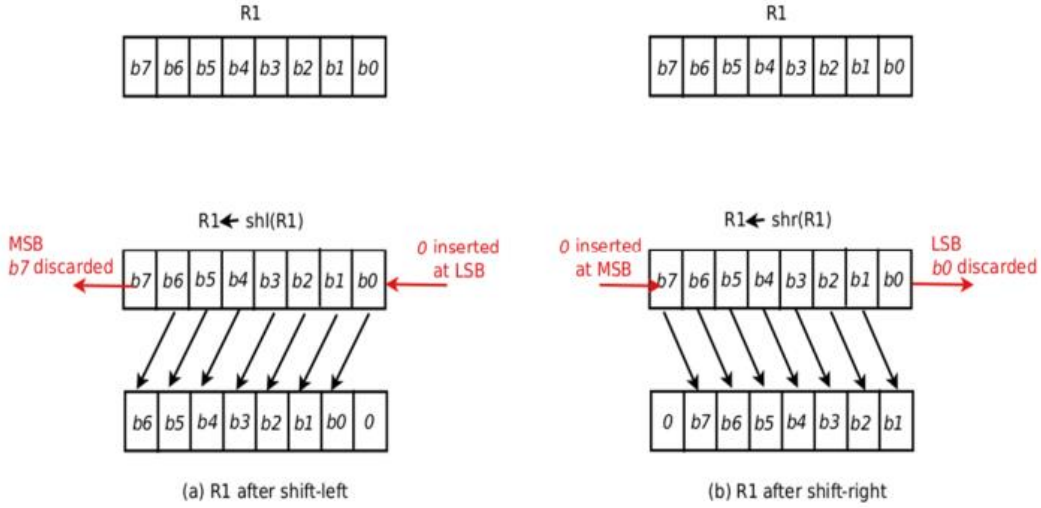
शिफ्ट मायक्रो-ऑपरेशन्स (Shift Micro-operations)

शिफ्ट मायक्रो-ऑपरेशन्सचा वापर करून माहिती क्रमाने ट्रान्सफर केली जाते. शिफ्ट माक्रो-ऑटोनाॅमसचा वापर करू शकत नाही, परंतु हे सत्य आहे.

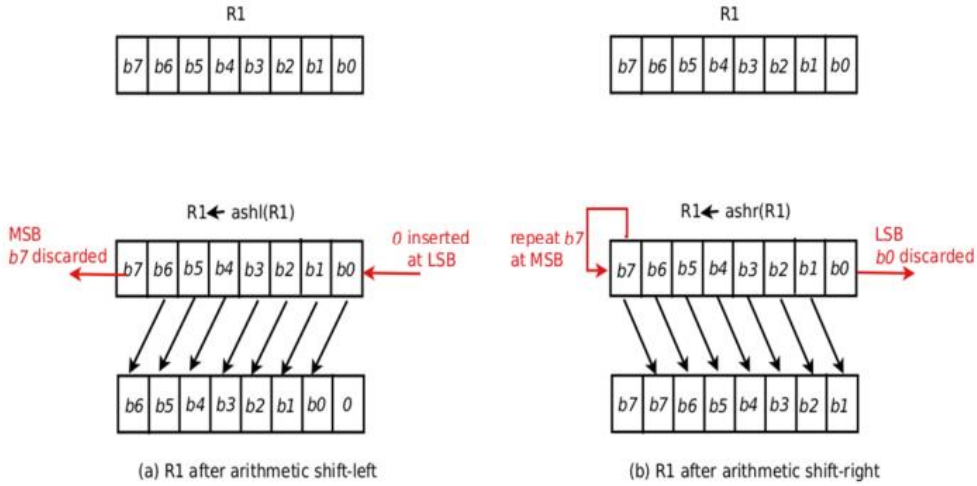
1. **लॉजिकल: सिरीयल इनपुट शून्य ऍंड्रेसतरीत करते.** "shl" आणि "shr" ही चिन्हे अनुक्रमे डावीकडे आणि उजवीकडे लॉजिकल बदलांसाठी वापरली जातात.

• **लॉजिकल शिफ्ट-लेफ्ट:** हे शिफ्ट लेफ्ट ऑपरेशन प्रत्येक बिट एक जागा लेफ्ट पोजिशन ला हलवते. आकृती 1.21 (अ) मध्ये एमएसबी डिसकार्ड केला आहे आणि एलएसबी शून्याने भरलेला आहे (the serial input).

- **लॉजिकल शिफ्ट-राइट:** याचा परिणाम आकृतीमध्ये दर्शविल्याप्रमाणे प्रत्येक बिटच्या उजवीकडे बदलण्यात होतो. 1.21 (b) मध्ये एलएसबी काढून टाकणे आणि रिक्त एमएसबी स्थितीत शून्य जोडणे.



आकृती. 1.21 लॉजिकल शिफ्ट (a) डावे आणि (b) उजवे सूक्ष्म ऑपरेशन



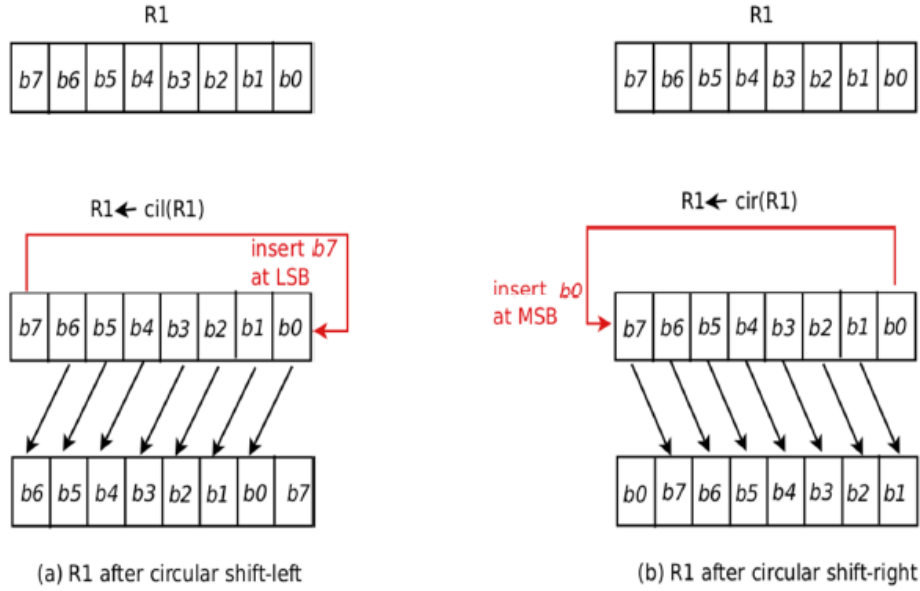
आकृती. 1.22 एरिथमेटिक शिफ्ट (a) डावीकडे आणि (b) उजवीकडे सूक्ष्म कार्ये

2. **अरीथमेटिक Arithmetic.** डावीकडील शिफ्ट आणि उजवीकडील शिफ्टला अरीथमेटिक शिफ्ट लेफ्ट आणि शिफ्ट राईट मायक्रो-ऑपरेशन्स म्हणून संबोधले जाते जेव्हा ते साइन बायनरी व्हॅल्यूवर लागू केले जातात. शिफ्ट लेफ्ट करत असताना बायनरी साइन पूर्णांकाचा 2 ने गुणाकार केला जातो. शिफ्ट राईट करत असताना ती संख्या 2 ने विभागली जाते.

- **अरीथमेटिक शिफ्ट लेफ्ट:** प्रत्येक बिट एका वेळी डावीकडे शिफ्ट केला जातो. एमएसबी टाकून दिला जातो आणि रिकामा एलएसबी शून्यावर सेट केला जातो. आकृती 1.22(a). मध्ये दर्शविल्याप्रमाणे हे डावीकडे केलेल्या लॉजिकल शिफ्टसारखेच आहे.

- **अरीथमेटिक शिफ्ट राईट:** एलएसबी टाकून दिला जातो, रिकामा एमएसबी मागील एमएसबी च्या व्हॅलूने भरलेला असतो आणि प्रत्येक बिट एका वेळी उजवीकडे शिफ्ट केला जातो. हे आकृती 1.22(b). मध्ये दर्शविलेले योग्य अरीथमेटिक शिफ्ट राईट म्हणून ओळखले जाते.

3. **सर्क्युलर (Circular) :** सर्क्युलर शिफ्ट कोणतीही माहिती न गमावता दोन्ही टोकांच्या भोवती रजिस्टरचा बिट क्रम पुन्हा फिरवते.



आकृती. 1.23: सर्क्युलर शिफ्ट (a) डावे आणि (b) उजवे मायक्रो ऑपरेशन्स

- **सर्क्युलर शिफ्ट-लेफ्ट:** आकृती 1.23(a). मध्ये दर्शविल्याप्रमाणे इतर सर्व बिट्स पुढच्या स्थानावर हलविताना एमएसबी बिट पहिल्या स्थानावर हलविला जातात.
- **सर्क्युलर शिफ्ट राईट:** एलएसबी ला एमएसबी किंवा शेवटच्या स्थानावर हलविले जाते, तर इतर सर्व बिट्स आकृती 1.23(b). मध्ये दर्शविल्याप्रमाणे मागील स्थानावर हलविले जातात.

युनिट सारांश

- संगणक हा इनपुट युनिट, आउटपुट युनिट, एरिथमेटिक लॉजिक युनिट (ALU.) मेमोरी युनिट आणि कंट्रोल युनिटने बनलेला असतो.
- डिजिटल संगणकांमध्ये वॉन न्युमन आर्किटेक्चरमध्ये प्रोग्राम आणि डेटा सेटयित करण्यासाठी एकच सामायिक मेमोरी असते आणि प्रोसेसरद्वारे मेमोरी प्रवेशासाठी एकच बस वापरली जातात.
- बायनरी अंक बस नावाच्या भौतिक तारांच्या जाळ्यामार्फत एका रजिस्टर/युनिटमधून इतर युनिट/रजिस्टरमध्ये पाठवले जातात.
- एमएआर(MAR), एमडीआर(MDR), पीसी(PC), आयआर(IR) आणि जनरल परपज रजिस्टर ही प्रोसेसरला आवश्यक असलेली विशिष्ट प्रकारची माहिती साठवतात.
- संगणक प्रणाली बायनरी संख्या (एकतर 0 किंवा 1) बिट्सची स्ट्रिंग म्हणून संग्रहित करते.
- इंटेजर नंबर साइन किंवा अन साइन असू शकतात. नंबर साइन चे, साइन आणि माग्निट्यूड, 1s कॉम्प्लिमेन्ट आणि 2s कॉम्प्लिमेन्ट संख्या प्रणालींमध्ये वर्गीकरण केले जातात.
- मायक्रो-ऑपरेशन्समधील समस्या टाळण्यासाठी संगणक साइन इंटेजर साठी 2s कॉम्प्लिमेन्ट प्रतिनिधित्व (-2^{n-1} ते $+2^{n-1}-1$ पर्यंतची व्हॅल्यू) वापरतात कारण साइन-अँड-माग्निट्यूडमध्ये 0 (+0 आणि -0) चे दोन प्रतिनिधित्व आहेत.
- जेव्हा n अंकी पूर्णांकांच्या बेरीजमध्ये n + 1 अंक असतात तेव्हा ओव्हरफ्लो होतो.
- संगणक फिक्स्ड पॉइंट आणि फ्लोटिंग पॉइंट क्रमांकाचे प्रतिनिधित्व वापरतात.
- प्राप्तकर्त्याने प्राप्त केलेली माहिती प्रेषकाने पाठवलेल्या माहितीशी जुळत नसल्यास त्रुटी उद्भवते.
- सिंगल बिटमधील बदल सिंगल-बिट एरर मानला जातो किंवा एकापेक्षा जास्त बिटमधील बदलाला बिटस्ट्रीम एरर म्हणतात. सम पॅरिटी कोड किंवा विषम पॅरिटी कोड त्रुटी शोध पद्धतींनी या त्रुटी शोधल्या जाऊ शकतात.
- सूक्ष्म कार्ये (मायक्रो ऑपरेशन) ही रजिस्टर वर केली जाणारी प्राथमिक कार्ये आहेत.
- रजिस्टर मधील डेटा सामायिक बसद्वारे पाठविला जातो ज्यामध्ये प्रत्येक बिटसाठी एक वायर, रेषा किंवा तारांची मालिका वापरली जातात.
- प्रत्येक रजिस्टर ट्रान्सफरदरम्यान कंट्रोल सिग्नल च्या आधारे बस डेस्टिनेशन रजिस्टर निवडते.
- विशिष्ट सूक्ष्म कार्ये करण्यासाठी हार्डवेअर लॉजिक सर्किट्सद्वारे रजिस्टर ट्रान्सफर सुलभ केले जातात.
- जेव्हा माहिती मेमोरी युनिटमधून अंतिम वापरकर्त्याकडे हलवली जाते तेव्हा मेमोरी ट्रान्सफरला रीड ऑपरेशन म्हणून संबोधले जाते आणि राइट ऑपरेशन म्हणजे मेमोरीमध्ये नवीन माहिती सेटयित करण्याची प्रक्रिया होय.

- अरीथमेटिक लॉजिक शिफ्ट युनिट (एएलएसयू) अरीथमेटिक, लॉजिक आणि शिफ्ट ऑपरेशन्स एकाच सर्किट मध्ये समाकलित करतात.
- अरीथमेटिक सूक्ष्म क्रिया म्हणजे बेरीज, वजाबाकी, 1s कॉम्पलिमेन्ट, 2s कॉम्पलिमेन्ट, वाढ आणि घट.
- लॉजिक मायक्रोऑपरेशन्स वैयक्तिक बिट्स हाताळू शकतात. हे ऑपरेशन बिट व्हालू बदलू, हटवू आणि समाविष्ट करू शकतात. AND, OR, XOR, NOT इ. सारख्या डिजिटल लॉजिक गेटचा वापर करून ही सूक्ष्म कार्ये अंमलात आणली जातात.
- मायक्रो-ऑपरेशन्स माहिती ला क्रमाने ट्रान्सफर करतात.
- या सूक्ष्म क्रियांचे वर्गीकरण लॉजिकल, अरीथमेटिक आणि सक्क्युलर शिफ्ट ऑपरेशन्स असे केले जातात.
- अरीथमेटिक आणि शिफ्ट सूक्ष्म-कार्ये करण्यासाठी गुणाकार आणि भागाकार अरीथमेटिक कार्ये वापरली जातात.

स्वाध्याय:

बहुपर्यायी प्रश्न:

प्रश्न 1.1 दशांश संख्या 485 ही बायनरी समतुल्य आहे?

(अ) 111100101 (ब) 111110111 (क) 101111111 (ड) 101110111

प्रश्न 1.2 किती बिट्स दोन बाइट्सच्या बरोबरीचे आहेत?

(अ) 4 (ब) 8 (क) 12 (ड) 16

प्रश्न 1.3 कोणते मेमोरी युनिट सर्वात वेगवान आहे?

(अ) रजिस्टर (ब) कॅशे (क) हार्डडिस्क (ड) रॅम

प्रश्न 1.4 पीसीचे व्हॅलू कसे बदलते?

(अ) वर्तमान रजिस्टर चे व्हॅलू (ब) मागील रजिस्टर (क) पुढील रजिस्टर ऍड्रेस (ड) या पैकी काही नाही

प्रश्न 1.5 आणायच्या सूचनेचा मेमोरी ऍड्रेस आणणे-डिकोड-एक्झिक्युट सायकल दरम्यान _ _ _ _ _
द्वारे प्रोग्राम काउंटरवरून RAM मध्ये ट्रान्सफर केला जातो.

(अ) कंट्रोल बस (ब) ऍड्रेस बस (क) डेटा बस (ड) एकतर (अ) किंवा (ब)

प्रश्न 1.6 रिकाम्या जागा भरण्यासाठी योग्य क्रम निवडा.

----- मध्ये डेटाचा मेमोरी ऍड्रेस असतो ज्यामध्ये CPU ने प्रवेश करणे आवश्यक आहे. CPU मध्ये _____ डेटा ठेवतो मेमोरी स्थानावर किंवा येथून ट्रान्सफर करत आहे. कार्यान्वित केल्या जाणाऱ्या पुढील डारेक्टिवचा मेमोरी ऍड्रेस ----- मध्ये संग्रहित केला जातो. एरिथमेटिक/ लॉजिक एकाद्वारे केलेल्या गणनेचा परिणाम ----- मध्ये संग्रहित केला जातो.

(अ) ACC, PC, MDR, MAR

(ब) MAR, PC, MDR, ACC

(क) PC, ACC, MDR, MAR

(ड) MAR, MDR, PC, ACC

प्रश्न 1.7 8-बिट वर्ड कॉम्प्युटरवरील शेवटचे ऑपरेशन म्हणजे $A = 11110000$ आणि $B = 00010100$ हे ऑपरेंड असलेले सबट्रॅक्शन (A-B) असे गृहीत धरले तर ओव्हरफ्लो, साइन, कॅरी, झिरो आणि इव्हन पॅरिटी फ्लॅगचे व्हॅल्यू किती असेल? [संकेत: 2 ची पूरक वजाबाकी पद्धत वापरा, झिरो फ्लॅग: जेव्हा एरिथमेटिक ऑपरेशनचा परिणाम शून्य असतो, तेव्हा झिरो फ्लॅग 1 वर सेट केला जातो, सम पॅरिटी फ्लॅग: निकालातील 1 ची संख्या सम असल्यास हा फ्लॅग सेट केला जातो].

(अ) 1, 0, 0, 0, 1

(ब) 0, 0, 0, 1, 1

(क) 0, 0, 0, 0, 1

(ड) 0, 1, 1, 0, 0

प्रश्न 1.8 4-बिट ALU हे साइन केलेल्या संख्यांसाठी $5 + (-5)$ ऑपरेशन करते असे गृहीत धरल्यास, कॅरी, ओव्हरफ्लो, झिरो, निगेटिव्ह आणि इव्हन पॅरिटी फ्लॅगची व्हॅल्यू काय आहेत? [संकेत: 2 चे पूरक एरिथमेटिक वापरा; नकारात्मक फ्लॅग: निकाल नकारात्मक असल्यास नकारात्मक फ्लॅग 1 वर सेट केला जातो, सम पॅरिटी फ्लॅग: निकालातील 1 ची संख्या सम असल्यास हा फ्लॅग सेट केला जातो]

(अ) 1, 1, 0, 1, 0

(ब) 1, 0, 1, 0, 1

(क) 1, 1, 1, 0, 1

(ड) 1, 0, 1, 1, 1

प्रश्न 1.9 फक्त आणि फक्त किमान एक इनपुट 1 असेल तर खालीलपैकी कोणते गेट आउटपुट 1 देते?

(अ) NOR

(ब) AND

(क) XOR

(ड) OR

प्रश्न 1.10 दोन इनपुटसह OR गेटसारखाच प्रभाव प्रदान करण्यासाठी दोन इनपुटसह किती NAND गेट आवश्यक आहेत?

(अ) चार

(ब) तीन

(क) दोन

(ड) एक

प्रश्न 1.11 शिफ्ट रजिस्टरचे योग्यरित्या असे रजिस्टर म्हणून वर्णन केले जाऊ शकते जे माहितीचे बिट्स बदलू शकते?

(अ) एकतर उजवीकडे किंवा डावीकडे (ब) फक्त डावीकडे (क) फक्त उजवीकडे (ड) दुसऱ्या रजिस्टर मध्ये

प्रश्न 1.12 एक्स म्हणजे 2 कॉम्पलिमेंट स्वरूपात 8-बिट पूर्णांकांच्या विशिष्ट संख्येचे प्रतिनिधित्व. चिन्ह आणि परिमाण प्रतिनिधित्वामध्ये Y हे 8-बिट पूर्णांकांचे विशिष्ट संख्येचे प्रतिनिधित्व असू द्या. तर, $X - Y = ?$

(अ) 1 (ब) 0 (क) 2 (ड) यापैकी काहीही नाही

प्रश्न 1.13 एक मल्टीप्लेक्सर

1. असंख्य इनपुटपैकी एक निवडा आणि ते एकाच आउटपुटवर पाठवा.
2. डेटा एकाच इनपुटमधून एकापेक्षा जास्त आउटपुटमध्ये पाठवला जातो.
3. पॅरलल डेटामधून सिरीयल डेटामध्ये रूपांतरित करते.
4. हे एक संयुक्त सर्किट आहे.

खालीलपैकी कोणता पर्याय खरा आहे हे ठरवा?

(अ) 1, 2, 4 (ब) 2, 3, 4 (क) 1, 3, 4 (ड) 1, 2, 3

प्रश्न 1.14 योग्य विधाने सर्वात अचूकपणे परिभाषित करणारा पर्याय निवडा.

1. एमएसबी मधील कॅरी आउट बिट व्हॅल्यू ओव्हरफ्लो दर्शवत नाही.
2. जर दोन्ही बेरीजमध्ये समान चिन्ह असेल तरच ओव्हरफ्लो शक्य आहे.
3. विरुद्ध चिन्हे असलेले पूर्णांक जोडताना अतिप्रवाह होऊ शकत नाही.
4. जर एमएसबी वर कॅरी इन आणि कॅरी आउटचा XOR 1 च्या बरोबरीचा असेल तर ओव्हरफ्लो आहे.

खालीलपैकी कोणता पर्याय खरा आहे हे ठरवा?

(अ) 1, 2, 4 (ब) 2, 3, 4 (क) 1, 3, 4 (ड) सर्व बरोबर आहेत

प्रश्न 1.15 मेमोरी अकॅसेस टाइम काय आहे?

(अ) ~100 ms (ब) ~1 ms (क) 10-30 ns (ड) 3-10 ns

बहुपर्यायी प्रश्नाचे उत्तरे

1.1	(अ)	1.2	(ड)	1.3	(अ)	1.4	(क)	1.5	(ब)	1.6	(ड)
1.7	(ड)	1.8	(क)	1.9	(ड)	1.10	(ब)	1.11	(अ)	1.12	(अ)
1.13	(क)	1.14	(ड)	1.15	(क)						

लघु आणि दीर्घ उत्तराचे प्रश्न

श्रेणी-I

प्रश्न 1.1: कॉम्प्युटरचे विविध भाग कोणते आहेत?

प्रश्न 1.2 कॉम्प्युटर ऑर्गनायझेशन आणि कॉम्प्युटर आर्किटेक्चरमध्ये काय फरक आहे?

प्रश्न 1.3 जेव्हा इंटररॅप्ट येतो तेव्हा प्रोग्राम काउंटरचे व्हॅल्यू कसे बदलते?

प्रश्न 1.4 कॅशे, प्रायमरी आणि सेकंडरी मेमोरी स्टोरेजमध्ये फरक करा.

प्रश्न 1.5: वॉन न्यूमन आर्किटेक्चर ची वैशिष्ट्ये सांगा.

प्रश्न 1.6 चिपवर प्रोसेसरची संख्या वाढवण्याच्या दृष्टीने कोणते इंटरकनेक्ट सर्वात कार्यक्षम आहे असे तुम्हाला वाटते?

प्रश्न 1.7 कंट्रोल लाइन चे कार्य काय आहे?

प्रश्न 1.8: पाच प्रमुख इनपुट उपकरणांची नावे सांगा?

प्रश्न 1.9 पाच वेगवेगळ्या प्रकारच्या आउटपुट उपकरणांची यादी करा.

प्रश्न 1.10 बस आणि मेमोरी ट्रान्सफर म्हणजे काय?

प्रश्न 1.11 ओव्हरफ्लो डिटेक्शनची कंडिशन काय आहे?

प्रश्न 1.12 फ्लोटिंग पॉइंट नंबर दर्शविण्यासाठी IEEE 754 मानक काय आहे?

प्रश्न 1.13 लॉजिक मायक्रो ऑपरेशन्स म्हणजे काय? वास्तविक जगात लॉजिक ऑपरेशन्सच्या तीन उपयोगांची यादी करा.

प्रश्न 1.14 सूक्ष्म कार्याचे विविध प्रकार कोणते आहेत? शिफ्ट मायक्रो-ऑपरेशन्स कधी कराव्यात?

श्रेणी-II

प्रश्न 1.15 एकाच वेळी एक्स. ओ. आर., एक्सक्लुझिव्ह-एन. ओ. आर., एन. ओ. आर. आणि एन. ए. एन. डी. सारख्या लॉजिक ऑपरेशन्स करण्यासाठी डिजिटल सर्किट तयार करा. आउटपुट रेपेवरील विशिष्ट गेटचे आउटपुट मिळविण्यासाठी दोन सिलेक्शन व्हेरिएबल्स वापरणे आवश्यक आहे. दर्शविण्यासाठी लॉजिक आकृती दाखवा.

प्रश्न 1.16 हे सिद्ध करा की XOR फंक्शन $x = \alpha \oplus \beta \oplus \rho \oplus \omega$ हे एक विषम फंक्शन आहे. फक्त $x = 1$ असेल तरच α, β, λ आणि ω मध्ये फक्त विषम संख्या आहेत हे दर्शवा.

प्रश्न 1.17 3-बिट पॅरिटी जनरेटर आणि 4-बिट पॅरिटी चेकरसाठी डिजिटल सर्किट तयार करा जे ऑड-पॅरिटी बिट वापरते.

प्रश्न 1.18 संगणकाला फ्लोटिंग पॉइंट नंबर रिप्रेझेंटेशनची आवश्यकता का आहे? संगणक या संख्यांचे प्रतिनिधित्व कसे करतात? फ्लोटिंग पॉइंट नंबर नॉर्मलायझेशनची व्याख्या काय आहे? सामान्यीकरण महत्वाचे का आहे? फ्लोटिंग पॉइंट रिप्रेझेंटेशनच्या विविध मानकांचे वर्णन करा.

संख्यात्मक समस्या Numerical Problems

प्रश्न 1.19 बायनरी व्हॅल्यू (101110) 2, (1110101) 2, आणि (110110100) 2 दशांश मध्ये रूपांतरित करा.

[Ans: 46, 117, 436]

प्रश्न 1.20 खालील 8-अंकी बायनरी संख्या आहेत; त्यांचे 1 आणि 2 चे पूरक शोधा: 00000000; 00000001; 10101110; 10000000; आणि 10000001.

[Ans: 1's complement: 11111111, 11111110, 01010001, 01111111, 01111110]

2's complement: 00000000, 11111111, 01010010, 10000000, 01111111]

प्रश्न 1.21 सबट्राहेंडचे 2 कॉम्प्लिमेंट वजा करून खालील साइन न केलेल्या बायनरी व्हॅल्यूवर वजाबाकी करा.

(i) 100 - 110000 (ii) 11010 - 1101

(iii) 11010 - 10000 (iv) 1010100 - 1010100

[Ans: (i) 010100 (ii) 01101 (iii) 01010 (iv) 0000000]

प्रश्न 1.22 रजिस्टर A मध्ये बायनरी अंक 11011001 आहेत. A चे व्हॅल्यू बदलण्यासाठी B ऑपरेंड आणि आवश्यक लॉजिक मायक्रो-ऑपरेशनची गणना करा:

(i) 11111101 (ii) 01101101

[Ans: B = 11111101, OR (ii) B = 10110100, XOR]

प्रश्न 1.23 2 कॉम्प्लिमेंट च्या संख्या प्रणालीचा वापर करून तुम्ही खालील साइन केलेल्या संख्यांची गणना कशी करता?

(i) (+70) + (+80)

(ii) $(-70) + (-80)$

(iii) $(+42) + (-13)$

(iv) $(-42) - (-13)$

प्रत्येक संख्येला त्याच्या चिन्हासह बायनरी व्हॅल्यू सेटयित करण्यासाठी आठ बिट्स असतात. एरिथमेटिक क्रियांचे परिणाम काय असतील? ओव्हरफ्लो आहे का?

[Ans: Overflow occurs only in the first two cases, (i) 10010110 (ii) 01101010
(iii) 00011101 (iv) 11100011]

प्रश्न 1.24 8-बिट रजिस्टर R1, R2, R3 आणि R4 ची प्रारंभिक व्हॅल्यू खालीलप्रमाणे आहेत: R1 = 11110010, R2 = 11111111, R3 = 10111001, and R4 = 11101010.

सूक्ष्म क्रियांचा खालील क्रम अंमलात आल्यानंतर प्रत्येक रजिस्टरमधील आठ बिटचे परिणाम फिक्स्ड करा.

R1 \leftarrow R1 + R2 Add R2 to R1

R3 \leftarrow R3 \wedge R4 AND R4 to R3

R2 \leftarrow R2 + 1 Increment R2

R1 \leftarrow R1 - R3 Subtract R3 from R1

[Ans: R1 = 01001001; R2 = 00000000; R3 = 10101000; R4 = 11101010]

प्रश्न 1.25 बायनरी व्हॅल्यू 10011100 हे 8-बिट रजिस्टरमध्ये साठवले जाते. अरीथमेटिक शिफ्ट बरोबर गृहीत धरल्यास, रजिस्टरमधील नवीन व्हॅल्यू काय आहे? 10011100 च्या सुरुवातीच्या व्हॅल्यूपासून, अरीथमेटिक शिफ्ट नंतर रजिस्टरचे व्हॅल्यू शोधा आणि ओव्हरफ्लो आहे की नाही ते दर्शवा.

[Ans: arithmetic shift-right: 11001110, arithmetic shift-left: 00111000, overflow detected]

प्रश्न 1.26 R = 11111111 पासून सुरू होणारी लॉजिकल शिफ्ट-उजवीकडे, सकूलर शिफ्ट उजवीकडे, लॉजिकल शिफ्ट डावीकडे आणि सकूलर शिफ्ट डावीकडे, त्यानंतर रजिस्टर R मध्ये बायनरी व्हॅल्यूचा क्रम शोधा.

[Ans: लॉजिकल शिफ्ट-उजवीकडे: 01111111, सकूलर शिफ्ट उजवीकडे: 10111111,
लॉजिकल शिफ्ट डावीकडे: 01111110, सकूलर शिफ्ट डावीकडे: 11111100]

प्रश्न 1.27 8 बिट्सचा घातांक आणि एक चिन्ह बिट आणि 26 बिट्सचा मान्दिसा आणि एक चिन्ह बिट मिळून 36-बिट फ्लोटिंग-पॉइंट बायनरी पूर्णांक तयार होतो. मंडिसाचे सामान्यीकरण केले जाते. मान्दिसा आणि घातांक या दोन्हीसाठी चिन्ह आणि परिमाण वापरले जातात. जर शून्याकडे दुर्लक्ष केले गेले, तर लिहिलेले सर्वात मोठे आणि सर्वात लहान सकारात्मक पूर्णांक कोणते आहेत?

[Ans: Largest: $(1-2^{-26}) \times 2^{+255}$, Smallest: $-(1-2^{-26}) \times 2^{+255}$]

प्रश्न 1.28 24-बिट बायनरी फ्लोटिंग पॉइंट रिप्रेझेंटेशन (binary floating point representation) + 46.5 म्हणजे काय? मान्दिसाचे सामान्यीकरण केले जाते आणि मान्दिस आणि घातांक अनुक्रमे 16 आणि 8 बिट्सद्वारे दर्शविले जातात.

[Ans: Binary representation: $(101110.1)_2$, Sign: 0, Mantissa: 1000010000000000, Exponent: 01111010]



स्कॅन करा

Xilinx ISE रचना संच
स्थापनेच्या स्टेप जाणून
घेण्यासाठी

प्रात्यक्षिक (PRACTICAL)

उद्देश: व्हेरिलॉग हार्डवेअर वर्णन लॅंग्वेज वापरून एरिथमेटिक लॉजिक शिफ्ट युनिटची लॉजिक आकृती काढा.
साधने: Xilinx ISE डिझाइन सूट [4]

सिद्धांत: व्हेरिलॉग मॉड्यूल हे मूलभूत वर्णनात्मक एकक आहेत. मॉड्यूल ते घोषित करते आणि एंड मॉड्यूल नेहमी ते समाप्त करते. मॉड्यूलसला नावे आणि पोर्ट असतात. मॉड्यूलचे नाव अर्थपूर्ण असावे. वर्णमाला, अल्फान्यूमेरिक आणि अंडरस्कोर नावे केस संवेदनशील आहेत. नावाचे पहिले अक्षर एकतर वर्णमाला वर्ण किंवा अंडरस्कोर असणे आवश्यक आहे. क्रमांकापासून नाव सुरू करण्याचा कोणताही मार्ग नाही.



स्कॅन करा

डिजिटल सर्किट्ससाठी
व्हेरिलॉग शिकण्यासाठी

पोर्ट त्यांच्या वातावरणासाठी इंटरफेस मॉड्यूलसची यादी करते. पोर्ट्स हे सर्किटचे इनपुट आणि आउटपुट असतात. वातावरण सर्किटची इनपुट लॉजिक व्हॅल्यू निर्धारित करते, तर सर्किटची आउटपुट लॉजिक व्हॅल्यू इनपुटद्वारे निर्धारित केली जातात. पोर्ट यादी कोष्ठकामध्ये जोडलेली असते जी अल्पविरामाद्वारे विभक्त केली जाते. वाक्यांनंतर अर्धविराम असतात, परंतु एंडमॉड्यूलस नसतात. प्रत्येक व्हेरिएबलचे सीमांकन करण्यासाठी अल्पविराम वापरले जातात. मुख्य शब्दांसाठी फक्त लहान अक्षरांचा वापर करा. त्यानंतर, इनपुट आणि आउटपुट अनुक्रमे इनपुट आणि आउटपुट पोर्टचा संदर्भ घेतील. अंतर्गत जोडण्या तारांच्या उपस्थितीद्वारे दर्शविल्या जातात.

परिपथ बांधणी "AND", "NOT", "OR" इत्यादी वर्णनात्मक आदिम द्वारांच्या यादीने निर्दिष्ट केली जाते. गेट इन्स्टन्स हे यादीचे घटक आहेत. प्रत्येक गेट त्वरित एका पर्यायी नावाने (गेट 1, गेट 2 इ.) सुरू होते. आणि गेट आउटपुट आणि इनपुट, अल्पविरामाद्वारे वेगळे केले आणि कंसात गुंडाळले. मूलभूत गेटचे इनपुट त्याच्या आउटपुटचे अनुसरण करते.

प्रिमिटिव्हच्या आउटपुटचा प्रथम उल्लेख करणे आवश्यक आहे, तर मॉड्यूलचे इनपुट आणि आउटपुट कोणत्याही क्रमाने निर्दिष्ट केले जाऊ शकतात. मॉड्यूलचे वर्णन एंड मॉड्यूल या वर्डने संपते.

प्रक्रिया:

1. प्रथम, सर्किटच्या इनपुट आणि आउटपुटची संख्या पहा. तुमच्या परिपथासाठी चांगले नाव निवडा. सर्किट _ नेम या कोड मॉड्यूलने रिक्त जागा भरा (list of input and output ports separated by comma) (verilog program here) अंतिम मॉड्यूल.



स्कॅन करा

Xilinx मध्ये testbench
तयार करणे आणि ISim
चालवणे शिकण्यासाठी

2. इनपुट व्हेरिएबल्स इनपुट व्हेरिएबल नावे म्हणून घोषित करा;
3. आउटपुट पोर्टची नावे म्हणून आउटपुट घोषित करा;
4. एका वायरला इंटरमिजिएट आउटपुट म्हणून नियुक्त करा जे दुसऱ्या सर्किटचे इनपुट बनेल.
5. परिभाषित मूलभूत गेटची यादी, ज्यापैकी प्रत्येक मुख्य वर्डद्वारे वैशिष्ट्यीकृत आहे, सर्किट डिझाइन परिभाषित करण्यासाठी वापरली जाते जसे की आणि, नाही, किंवा, इ.
6. यादीतील प्रत्येक घटक एक गेट उदाहरण आहे.
7. गेटचे प्रत्येक उदाहरण गेट 1, गेट 2 इ. सारख्या नावाने किंवा लेबलने सुरू होते. त्यानंतर, गेटचे आउटपुट आणि इनपुट नंतर अल्पविराम आणि कंस वापरून वर्णन केले जातात.
8. आदिम गेटच्या आउटपुटचा नेहमी प्रथम उल्लेख केला जातो, त्यानंतर इनपुटचा.
9. परिपथ पडताळणीसाठी चाचणीपीठ तयार करा. सर्व संभाव्य इनपुट आणि आउटपुट संयोजनांचा उल्लेख करा.
10. आउटपुट पाहण्यासाठी आणि सर्किटचे कार्य तपासण्यासाठी Xilinx मध्ये Isim चालवा.

अधिक जाणून घ्या **KNOW MORE**

भारतीयांनी केलेले नवकल्पना

भारतीय लोक वारंवार जग बदलण्याची किंवा किमान जगण्यासाठी एक चांगले ठिकाण बनवण्याची क्षमता दर्शवतात. युएसबी (युनिव्हर्सल सिरियल बस) हा आपल्या दैनंदिन जीवनाचा अविभाज्य भाग आहे. या कॉम्पॅक्ट डेटा स्टोरेज डिव्हाइसचा सह-शोध भारतीय-अमेरिकन संगणक वास्तुविशारद अजय भट्ट यांनी लावला होता. मात्र, या शोधामुळे त्याला पैसे मिळाले नाहीत. त्यांनी हे आर्थिक फायद्यासाठी केले नाही, तर तंत्रज्ञानात बदल घडवून आणण्यासाठी केले. त्यांच्या मते, असा महत्त्वपूर्ण बदल घडवून आणण्याची संधी क्वचितच मिळते. दरवर्षी 11 मे रोजी भारत भारतीय तंत्रज्ञान दिन साजरा करतो आणि तरुण शोधकांना पुरस्कार देतात.

भारतीय वैदिक विज्ञान

वैदिक शिक्षणात मानवी ज्ञानाचा अफाट खजिना आहे, जो हजारो वर्षांपासून आपल्यासोबत आहे. मौखिक परंपरांमध्ये अस्तित्वात असलेले भारताचे वैदिक विज्ञान तसेच डझनभर भारतीय लिपी आणि भाषांमध्ये रजिस्टरवलेल्या 40 लाखांहून अधिक हस्तलिखिते आपल्याला आपले संगणकीय कौशल्य सुधारण्यासाठी एक दृढ व्यासपीठ प्रदान करू शकतात. उदाहरणार्थ, आधुनिक भाषेतील ग्रंथांपेक्षा शास्त्रातील ग्रंथ यंत्रप्रोसीजरसाठी लक्षणीयरीत्या अधिक योग्य आहेत. परिणामी, आजच्या शक्तिशाली संगणकीय तंत्रज्ञानामध्ये शिकण्याच्या वैदिक संकल्पनांचा वापर केल्याने ज्ञानाचा समृद्ध खजिना मिळतो जो दोन्ही जगातील सर्वोत्तम गोष्टींचे मिश्रण करतो.

शून्य, द्वैती संख्या प्रणाली, बीजगणितीय परिवर्तन, हॅशिंग, पुनरावृत्ती, औपचारिक व्याकरण, गणितीय लॉजिक आणि उच्च स्तरीय भाषेचे वर्णन या सर्व कल्पना भारतात उद्भवल्या. 'जादुई सूत्रांचे' वेद म्हणून व्यापकपणे ओळखल्या जाणाऱ्या अथर्ववेदाने गणित सोपे केले आहे. सर्व वर्तमान संगणकांमध्ये वापरली जाणारी द्वैती योजना अथर्ववेदातून प्राप्त झाली आहे. वैदिक गणिताच्या वापराने, तास लागणारी गणना काही मिनिटांत पूर्ण केली जाऊ शकते.

आपण नकळत वैदिक सूत्रांचा वापर करतो. जेव्हा अनुक्रमे $i \leftarrow i+1$ आणि $i \leftarrow i-1$ वापरले जातात, तेव्हा सॉफ्टवेअर पद्धतींमध्ये 'एकादिकिन-पूर्वेण' आणि 'एकान्युनेना-पूर्वेण' सूत्रांचा वापर केला जातो. त्याचप्रमाणे, संगणक गुणक एककांमध्ये असंख्य वैदिक सूत्रे वापरली जातात [6]. हे जलद परिणाम देईल, जे क्रिप्टोग्राफी पद्धती आणि प्रतिमा प्रक्रिया अनुप्रयोगांसारख्या अनेक अनुप्रयोगांमध्ये आवश्यक आहे.

येथे प्राचीन वैदिक ज्ञानाचा समावेश करून संगणक विज्ञानाचा अधिक व्यापक आणि अधिक व्यापक दृष्टीकोन स्थापित करणे शक्य आहे. वैदिक संकल्पना नैसर्गिक नियमातून उद्भवल्या आहेत, ज्या ऋग्वेदात पुरेशा प्रमाणात व्यक्त केल्या आहेत. वैदिक ऋषींनी या नियमाकडे विश्वाची खरी नियामक शक्ती म्हणून पाहिले आणि वैदिक देवता देखील या कायद्याच्या अधीन किंवा संरक्षक म्हणून पाहिल्या जातात. नदीचा प्रवाह, रात्र आणि सकाळची घटना यासारख्या नैसर्गिक घटनांचे वर्णन या नैसर्गिक नियमाच्या संदर्भात केले आहे. आता हे सिद्ध झाले आहे की बॅबिलोनियामध्ये देवाच्या संदर्भात नैसर्गिक कायद्याची संकल्पना अस्तित्वात होती.

वेदांचे असे चार विशाल संग्रह आहेत, ते म्हणजे (i) ऋग्वेद, यज्ञ आणि विधींच्या वेळी पठित केले जाणारे स्तोत्रांचे किंवा स्तोत्रांचे आणि प्रार्थनांचे (मंत्र) पुस्तक, (ii) सामवेद, स्वरांचे पुस्तक (समन) ज्यासाठी स्तोत्र गायले जातात, (iii) यज्ञात्मक सूत्रांचे पुस्तक यजुर्वेद आणि (iv) अथर्ववेद, जादूच्या सूत्रांचे पुस्तक [7].

संदर्भ आणि सुचविलेले वाचन

[1] एम. मॉरिस मनो, संगणक प्रणाली वास्तुकला. प्रेंटिस-हॉल, इंक., तिसरी आवृत्ती. <https://poojavaishnav.files.wordpress.com/2015/05/mano-m-m-computer-systemarchitecture.pdf>. पीडीएफ (last accessed: Aug 15, 2022)

[2] कार्ल हमाकर, इवोन्को ब्रॅनेसिक, सफवत झाकी आणि नरैग मंजिकियन, संगणक संस्था आणि एम्बेडेड सिस्टीम्स. माकग्रा-हिल उच्च शिक्षण, 2011.

प्रा. इंद्रनील सेनगुप्ता आणि प्रा. कमलिका दत्ता यांचा एनपीटीईएल अभ्यासक्रम, संगणक वास्तुकला आणि संघटना, आयआयटी खरगपूर, 2017.

<https://archive.nptel.ac.in/cours/106/105/106105163/> / (last accessed: Aug 15, 2022)

[4] Xilinx ISE डिझाइन सेट.

<https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive-ise.html> (last accessed: Sept. 20, 2022)

[5] राष्ट्रीय तंत्रज्ञान दिन सोहळ्याविषयी.

<https://economictimes.indiatimes.com/news/how-to/national-technology-day-why-is-it-celebrated-history-behind-it/articleshow/100171949.cms> (last accessed: Aug 15, 2022)

[6] एस. जैन आणि V.S. जगतप, वेदीक माथेमाटिक्स इन कॉम्प्युटर: अ सर्वे. इंटरनॅशनल जर्नल ऑफ कॉम्प्युटर सायन्स अँड इन्फॉर्मेशन टेक्नॉलॉजीज, 5 (6) पृष्ठे 7458-7459, 2014.

[7] अ कन्साइझ हिस्ट्री ऑफ सायन्स इन इंडिया, संपादन. डी. एम. बोस, एस. एन. सेन आणि बी. व्ही. सुब्बरायप्पा, 1971.

[8] जतिंद्र कुमार डेका, अर्णब सरकार आणि संतोष विश्वास यांचा एन. पी. टी. ई. एल. अभ्यासक्रम, संगणक संघटना आणि वास्तुकला: एक शैक्षणिक दृष्टीकोन, आय. आय. टी. गुवाहाटी, 2017.
<https://archive.nptel.ac.in/cours/106/103/1>

2. मायक्रो प्रोग्रॅम कंट्रोल

युनिटची वैशिष्ट्ये

या युनिट मध्ये खालील पैलूंवर चर्चा केली आहे:

- कंट्रोल युनिटची मेमोरी आणि ऍड्रेस क्रमनिर्धारण नियंत्रित करणे;
- कंट्रोल युनिट ची मूलभूत रचना;
- साइन इंटेजर नंबर चे अरीथमेटिक कार्ये;
- गुणाकार आणि भागाकार साठी विविध पद्धती;
- फ्लोटिंग पॉइंट नंबरसाठी अरीथमेटिक ऑपरेशन्स, म्हणजेच, बेरीज, वजाबाकी, गुणाकार, भागाकार;
- अरीथमेटिक आणि इंस्ट्रक्शन पाइपलाइन, हार्डवर्ड आणि त्यांचे उपाय;
- RISC पाइपलाइन आणि वेक्टर प्रोसेसिंग;
- अरे प्रोसेसर.

अधिक जिज्ञासा आणि सर्जनशीलता वाढवण्यासाठी आणि समस्या सोडवण्याची कौशल्ये वाढवण्याच्या उद्देशाने विषयांचे व्यावहारिक अनुप्रयोग सादर केले जातात. ब्लूम वर्गीकरणशास्त्राच्या खालच्या आणि वरच्या स्तरांनुसार दोन श्रेणींमध्ये चिन्हांकित केलेल्या मोठ्या संख्येने बहुपर्यायी प्रश्न आणि लहान कीन्वा दीर्घ उत्तराच्या प्रश्नांव्यतिरिक्त, युनिट संख्यात्मक समस्यांच्या स्वरूपात सराव असाइनमेंट, संदर्भाची यादी आणि सुचविलेले वाचन प्रदान करते. हे लक्षात घेणे महत्वाचे आहे की स्वारस्य असलेल्या विविध विषयांवरील अधिक माहितीसाठी स्कॅन केले जाऊ शकणारे अनेक QR कोड वेगवेगळ्या भागांमध्ये समाविष्ट केले गेले आहेत आणि आवश्यक डेटा प्राप्त करण्यासाठी वापरले जाऊ शकतात.

सामग्रीवर आधारित संबंधित प्रात्यक्षिक नंतर विषयावरील "अधिक जाणून घ्या" विभाग आहे. हा विभाग काळजीपूर्वक अशा प्रकारे तयार करण्यात आला आहे की त्यात असलेली पूरक माहिती पुस्तकाच्या वाचकांसाठी मौल्यवान असेल. हा विभाग प्रामुख्याने संगणक प्रणाली संघटना, भारतीय आयुर्वेदिक ज्ञान, इतिहास आणि आपल्या दैनंदिन जीवनात नैसर्गिक पद्धतींच्या माध्यमातून निरोगी आणि उत्साही राहण्याचे महत्त्व विकसित करण्यासाठी भारतीय नवसंशोधकांच्या योगदानावर लक्ष केंद्रित करतो.

तर्क (RATIONALE)

कंट्रोल युनिट प्रोसेसरच्या कृतींचे समन्वय साधतात. हे एएलयू (ALU) च्या इंस्ट्रक्शन अंमलात आणण्यासाठी आणि इतर घटकांच्या कृतींचे समन्वय साधण्यासाठी निर्देशित करतात. हा अध्याय कंट्रोल युनिटच्या मूलभूत संरचनेचे परीक्षण करतो आणि प्रोग्राम इंस्ट्रक्शन कशा आणल्या जातात, डीकोड केल्या जातात आणि अंमलात आणल्या जातात हे दर्शवितो. प्रोसेसिंग युनिटला CPU (central processing unit) म्हणून संबोधले जाते. आधुनिक संगणकांमध्ये अनेक प्रोसेसिंग युनिट अस्तित्वात आहेत. म्हणून, प्रोसेसर ही संज्ञा CPU पेक्षा अधिक योग्य आहे. संगणकाचे अरीथमेटिक तर्कशास्त्राच्या रचनेच्या अनेक मनोरंजक पैलूंचा परिचय करून देतात .

हा अध्याय साइन आणि माग्रीटूड आणि 2s कॉम्प्लिमेंट संख्या प्रणालीसाठी बेरीज, वजाबाकी, गुणाकार आणि भागाकार यासारख्या संगणकाच्या अरीथमेटिक कार्यांच्या विविध तंत्रांची तपासणी करतो. आयईईई (IEEE) मानकांनुसार फ्लोटिंग-पॉइंट क्रमांकाचे प्रतिनिधित्व आणि फ्लोटिंग पॉइंट क्रमांकांचे अरीथमेटिक कार्य आयोजित करण्याच्या पद्धतींचे विश्लेषण केले जाते. फ्लोटिंग पॉइंट नंबर्स ची बेरीज, वजाबाकी, गुणाकार आणि भागाकार करण्याच्या पद्धतींवर चर्चा केली आहे. उच्च कार्यक्षमता देण्यासाठी तांत्रिक प्रगतीसह प्रोसेसर संघटना बदलली आहे. उच्च कार्यक्षमता प्राप्त करण्यासाठी अनेक कार्यात्मक युनिट समांतर कार्य करतात. अशा प्रोसेसरमध्ये एक पाईपलाईन ची रचना केली असते ज्यामध्ये मागील इंस्ट्रक्शन पूर्ण होण्यापूर्वी इंस्ट्रक्शन चे एक्सिक्युशन केल्या जाते. या अध्यायात पाईपलाईनची रचना, संभाव्य हझार्ड आणि प्रस्तावित उपाययोजनांचा सर्वसमावेशक आढावा देण्यात आला आहे.

पूर्व-आवश्यकता (PRE-REQUISITES)

गणित: पूर्णांक आणि फ्लोटिंग पॉइंट संख्यांसह एरिथमेटिक कार्ये (Class X)

डिजिटल इलेक्ट्रॉनिक्स: संख्या प्रणाली आणि डिजिटल लॉजिक गेट्स (Polytechnic Engineering)

युनिट निष्पत्ती (UNIT OUTCOMES)

या युनिट च्या निष्पत्ती ची यादी खालीलप्रमाणे आहे:

U2-O1: कंट्रोल युनिट डिझाइनमध्ये कंट्रोल मेमोरी आणि एड्रेस सिक्वेंसिंग च्या भूमिकेचे वर्णन करणे.

U2-O2: पूर्णांक संख्यांसाठी बेरीज, वजाबाकी, गुणाकार, भागाकार च्या विविध पद्धतींचे वर्णन करणे.

U2-O3: फ्लोटिंग पॉइंट नंबरसाठी आयईईई (IEEE) स्टॅंडर्ड आणि अरीथमेटिक कार्ये स्पष्ट करणे.

U2-O4: अरीथमेटिक आणि इंस्ट्रक्शन पाइपलाईन, डेटा आणि कंट्रोल हझार्ड आणि त्यांचे उपाय स्पष्ट करणे.

U2-O5: RISC पाइपलाईन, वेक्टर प्रोसेसिंग आणि अरे प्रोसेसर समजावून घेणे.

युनिट -2 निष्पत्ती	निष्पत्ती सह अपेक्षित मापिंग				
	1- कमकुवत मापिंग, 2- मध्यम मापिंग, 3- मजबूत मापिंग				
	CO-1	CO-2	CO-3	CO-4	CO-5
U2-01	3	3	2	3	3
U2-02	3	2	3	3	2
U2-03	3	3	2	2	1
U2-04	3	3	2	3	3
U2-05	3	2	3	2	1

2.1 कंट्रोल मेमोरी

कंट्रोल युनिट संगणकातील सूक्ष्म क्रियांचे अनुक्रम सुरू करते. कंट्रोल युनिटची कार्यक्षमता हार्डवायर्ड किंवा मायक्रोप्रोग्रामिंगच्या कोणत्याही एका दृष्टिकोनातून अंमलात आणली जाऊ शकते. हार्डवायर्ड दृष्टीकोन हार्डवेअरचा वापर करून कंट्रोल सिग्नल निर्माण करतो. तर, मायक्रोप्रोग्रामिंग ही डिजिटल संगणकातील मायक्रोऑपरेशन सिकवेन्स नियंत्रित करण्याची आणखी एक पद्धत आहे.

मायक्रोप्रोग्राम केलेले कंट्रोल युनिट मायक्रोप्रोग्राम सेटयित करते. प्रत्येक मायक्रोप्रोग्रामिंग सूक्ष्म इंस्ट्रक्शन किंवा कंट्रोल वर्ड चा क्रम असतो. हे कंट्रोल वर्ड बायनरी व्हॅल्यूद्वारे दर्शविले जातात, जे 1s आणि 0s च्या स्ट्रिंग आहेत. प्रत्येक सूक्ष्म इंस्ट्रक्शन किंवा कंट्रोल वर्ड मध्ये एक अद्वितीय बिट नमुना असतो. कंट्रोल युनिटची प्रत्येक बायनरी स्थिती मायक्रोऑपरेशन निर्दिष्ट करते आणि परिणामी कंट्रोल सिग्नल तयार होतात. परिणामी, कंट्रोल वर्ड प्रणाली घटकांवर वेगवेगळ्या क्रिया करण्यासाठी प्रोग्राम केले जाऊ शकतात. सूक्ष्म-रूटीन हा सूक्ष्म इंस्ट्रक्शन च्या मालिकेने बनलेला असतो.

एकदा कंट्रोल युनिट कार्यान्वित झाले की, पुढील मायक्रोप्रोग्राम अद्यावत करण्याची आवश्यकता नाही, कारण कंट्रोल मेमोरी (ROM) आहे. जेव्हा हार्डवेअर तयार केली जात असते, तेव्हा रॉम (ROM) मध्ये वर्ड कायमस्वरूपी बनवले जातात. ROM मधील वर्ड मायक्रोइन्स्ट्रक्शन्स साठी आहेत. कंट्रोल युनिट मध्ये कंट्रोल मेमोरी समाविष्ट असतात. कंट्रोल मेमोरी ही मुख्य मेमोरी सारखी नसते. कंट्रोल मेमोरीच्या उलट, प्रोग्राम मुख्य मेमोरी मध्ये बदलतो.

मशीन इंस्ट्रक्शन कंट्रोल मेमोरी मधील सूक्ष्म इंस्ट्रक्शन सुरू करतात. मायक्रोइन्स्ट्रक्शन्स मुख्य मेमोरीतून आणणे, प्रभावी ऍड्रेसचे व्हॅल्यूकन करणे, ऑपरेशन कार्यान्वित करणे आणि पुढील इंस्ट्रक्शन साठी सायकलची पुनरावृत्ती करण्यासाठी आणण्याच्या टप्प्यावर परत येणे यासारख्या सूक्ष्म कार्यांसाठी कंट्रोल सिग्नल तयार करतात.

डायनॅमिक मायक्रोप्रोग्रामिंगच्या वापराद्वारे, मायक्रो-प्रोग्राम चुंबकीय डिस्कसारख्या सहाय्यक मेमोरीतून डेटा लोड करण्यास सक्षम आहे. डायनॅमिक मायक्रोप्रोग्रामिंग लिहिण्यायोग्य कंट्रोल मेमोरी वापरते. ही मेमोरी सहसा रीड करण्यासाठी वापरली जाते परंतु मायक्रो-प्रोग्राममध्ये बदल करू शकते.

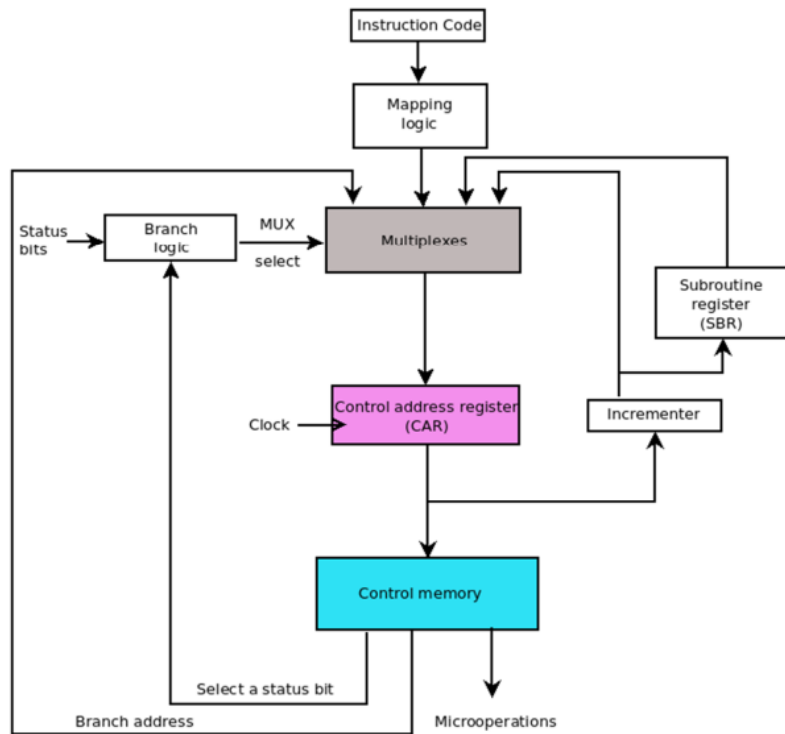
2.2 ऍड्रेस सिकवेन्सिंग

कंट्रोल मेमोरी सूक्ष्म इंस्ट्रक्शन चा संग्रह ठेवते आणि प्रत्येक संग्रह एक विशिष्ट रूटीन परिभाषित करतात. इंस्ट्रक्शन अंमलात आणण्यासाठी आवश्यक सूक्ष्म कार्ये तयार करण्यासाठी संगणक कंट्रोल मेमोरी मध्ये साठवलेल्या समर्पित सूक्ष्म-प्रोग्राम रूटीन वर अवलंबून असतो. कंट्रोल मेमोरी चा ऍड्रेस क्रम व्यवस्थापित करणारे हार्ड वेअर नित्यक्रमामध्ये सूक्ष्म इंस्ट्रक्शन क्रमबद्ध करण्यास आणि नित्यक्रमा मध्ये ब्रँच करण्यास सक्षम असणे आवश्यक आहे.

संगणकाचे कंट्रोल ऍड्रेस रजिस्टर जेव्हा प्रथम चालू केले जाते तेव्हा ते "आरंभीकृत" केले जाते. इन्स्ट्रक्शन फेच (fetch) सुरु करणारा मायक्रोइन्स्ट्रक्शन सिकवेन्स सहसा येथे सुरु होतो. जेव्हा कंट्रोल ऍड्रेस रजिस्टर वाढवली जाते तेव्हा फेच (fetch) करण्याचा नित्यक्रम तयार करणारे मायक्रोइन्स्ट्रक्शन क्रमबद्ध केले जातात. त्यानंतर इन्स्ट्रक्शन फेच (fetch) करण्याच्या टप्प्यानंतर संगणकाच्या इन्स्ट्रक्शन रजिस्टर (IR) वर पोहोचवली जाते.

कंट्रोल मेमोरी रूटीन ऑपरेंडचा प्रभावी ऍड्रेस ठरवते. संगणकाच्या इन्स्ट्रक्शन अप्रत्यक्ष ऍड्रेस आणि अनुक्रमणिका रजिस्टर पद्धती परिभाषित करू शकतात. ब्रांच मायक्रोइन्स्ट्रक्शन मोड बिट्सवर आधारित कंट्रोल मेमोरी इफेक्टिव्ह ऍड्रेस कॉम्प्युटेशन रूटीनमध्ये प्रवेश करू शकते. प्रभावी ऍड्रेसच्या गणनेनंतर मेमोरी ऍड्रेसची रजिस्टर ऑपरेंडचा ऍड्रेस सेटयित करेल.

मेमोरी इन्स्ट्रक्शन फेच (fetch) एक्सिक्युट करण्यासाठी कार्ये तयार करतात. ऑपरेशन कोड प्रोसेसर रजिस्टर च्या मायक्रोऑपरेशन चे स्टेजेस निर्धारित करतो. प्रत्येक इन्स्ट्रक्शनमध्ये एक मायक्रोप्रोग्राम असतो जो कंट्रोल मेमोरी मध्ये साठवला जातो. जेथे नित्यक्रम अस्तित्वात आहे, तेथे डारेक्टिव कोड बिट्स कंट्रोल मेमोरी मध्ये मॅप केले जातात. आवश्यक फेच रूटीन पर्यंत पोहोचल्यानंतर, कंट्रोल ऍड्रेस रजिस्टरमध्ये वाढ केल्याने इन्स्ट्रक्शन अंमलात आणणाऱ्या मायक्रोइन्स्ट्रक्शन्सचा क्रम लावता येतात, आणि प्रोसेसर रजिस्टर स्टेटस बिट्स मायक्रो ऑपरेशन सिक्वेन्सिंगवर देखील परिणाम करू शकतात.



आकृती. 2.1: कंट्रोल मेमोरी ऍड्रेस सिलेक्शन

सब-रूटीन आधारित मायक्रोप्रोग्राम ने रिटर्न ऍड्रेस बाह्य रजिस्टर मध्ये संचयित करणे आवश्यक आहे. कारण डिवाइस, रॉम (ROM) वर रिटर्न ऍड्रेस लिहू शकत नाही. इन्स्ट्रक्शन पूर्णपणे एक्सिक्युट झाल्या नंतर, कंट्रोल फेच

रुटीनमध्ये परत येणे आवश्यक आहे. फेच रुटीनच्या प्रारंभिक ऍड्रेसवर बिनशर्त ब्रांच करणारी सूक्ष्म इंस्ट्रक्शन कार्यान्वित होतात.

आकृती 2.1 मध्ये मायक्रोइंस्ट्रक्शन एड्रेस सर्किटरी दर्शवते. कंट्रोल मेमोरी मायक्रोइंस्ट्रक्शन्स संगणक रजिस्टर मायक्रोऑपरेशन्स सुरू करतात आणि पुढील ऍड्रेस कसा मिळवायचा ते परिभाषित करतात. कंट्रोल एड्रेस रजिस्टर (CAR) ला ऍड्रेस मिळण्याचे चार मार्ग प्रतिमेत दर्शवीले आहेत. इन्क्रिमेंटर कंट्रोल ऍड्रेस रजिस्टर वाढवून पुढील मायक्रोइंस्ट्रक्शन निवडतो. मायक्रोइंस्ट्रक्शन्स चे फील्ड ब्रांच चा ऍड्रेस परिभाषित करते. मायक्रोइंस्ट्रक्शनमधून स्टेटस बिट निवडून कंडिशनल ब्रांच सक्षम केली जाते. मारपींग लॉजिक सर्किट्स मेमोरी नियंत्रित करण्यासाठी बाह्य अँड्रेसिंग रिलोकेट करतात. मायक्रोप्रोग्रॅम त्याचा रिटर्न अँड्रेस वापरून सबरुटीनमधून बाहेर पडण्यासाठी विशिष्ट रजिस्टरचा वापर करतात.

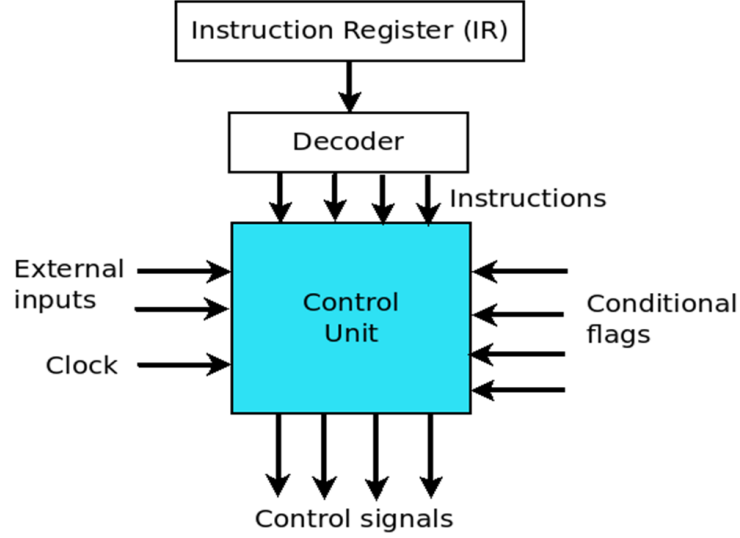
2.3 कंट्रोल युनिट डीजाइन

आकृती 2.2 मध्ये दर्शविल्याप्रमाणे रजिस्टर, अंतर्गत बस, एएलयू आणि प्रोसेसरमधील विविध घटक आणी या विविध घटकांमधील मार्ग सक्रिय करण्यासाठी कंट्रोल युनिट कंट्रोल सिग्नल तयार करतात. कंडिशन कोड, इन्स्ट्रक्शन रजिस्टर, ऑपरेशन कोड, बाह्य इनपुट आणि क्लॉक हे कंट्रोल युनिटसाठीचे इनपुट आहेत. हे इनपुट खालीलप्रमाणे विविध माहिती देतात.

1. इन्स्ट्रक्शन रजिस्टर (IR) मेमोरी डेटा रजिस्टर (MDR) मधून कार्यान्वित करण्यासाठी इन्स्ट्रक्शन लोड करतात. मेमोरी अँड्रेस रजिस्टर (MDR) द्वारे निर्दिष्ट केलेल्या मेमोरी मध्ये स्थानातील व्हॅल्यूची प्रत असते. डारेक्टिव डीकोड केल्यानंतर, प्रोसेसरला कोणते ऑपरेशन करायचे आहे हे माहित असते. हे ऑपरेशन बिट्स कंट्रोल युनिट साठी इनपुट आहेत.
2. कंडिशन कोड (फ्लॅग) बिट्स हे कंट्रोल युनिटसाठी इनपुट असतात. कंडिशन कोड मागील इन्स्ट्रक्शन ऑपरेशनची स्थिती देतात. उदाहरणार्थ, इन्क्रिमेंट अँड स्किपमध्ये जर झिरो (ISZ) इन्स्ट्रक्शन इफेक्टिव्ह अँड्रेस एकाने वाढवला असेल आणि झिरो बरोबर तुलना केली असेल तर झिरो फ्लॅग (ZF) 1 वर सेट केला जाईल आणि प्रोसेसर पुढील इन्स्ट्रक्शन एक्झिक्युशन वगळेल.
3. बाह्य इनपुट हे असे कंट्रोल संकेत आहेत जे प्रणाली बसमधून प्रदान केले जातात जसे की वेट फॉर मेमोरी फंक्शन कंप्लिट (WMFC) हे कंट्रोल संकेत मुख्य मेमोरीमधून तयार केले जातात. इनपुट उपकरणांमधून, कंट्रोल युनिटला इंटरप्ट (interrupt) विनंती संकेत देखील तयार केले जातात.
4. क्लॉक: प्रोसेसरमध्ये एकाच क्लॉक सायकल मध्ये एकाच वेळी अनेक सूक्ष्म कार्ये केली जातात.

2.3.1 हार्डवायर्ड कंट्रोल युनिट

लॉजिक गेट्स, फ्लिप-फ्लॉप, डीकोडर्स आणि इतर डिजिटल सर्किट्स हार्डवायर्ड कंट्रोल स्ट्रक्चरमध्ये कंट्रोल लॉजिकची अंमलबजावणी करतात. आरआयएससी (RISC) -आधारित संगणकांना हार्डवायर्ड नियंत्रणाची आवश्यकता असते. हार्डवायर्ड कंट्रोल स्ट्रक्चरची ब्लॉक डायग्रॅम आकृती 2.3 मध्ये दर्शविली आहे.



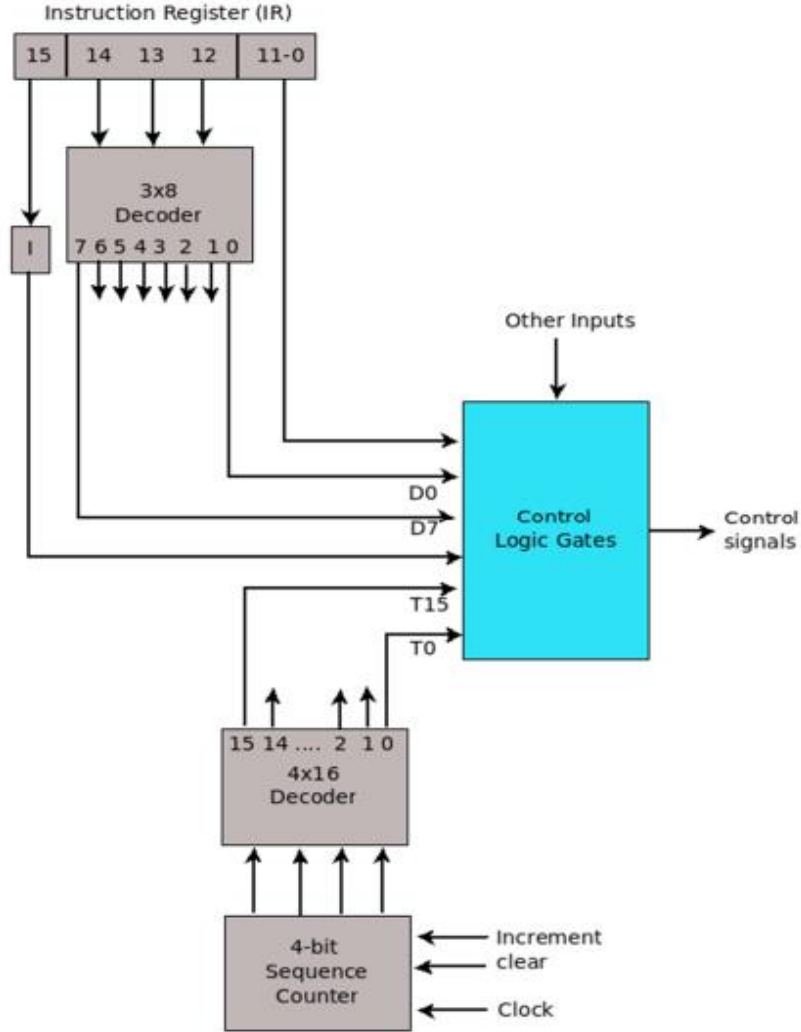
आकृती. 2.2: कंट्रोल युनिट ब्लॉक डायग्रॅम

दोन डिकोडर, एक सिकवेन्स काउंटर आणि अनेक लॉजिक गेट्स एकत्र करून हार्डवायर्ड कंट्रोल तयार करतात. इन्स्ट्रक्शन रजिस्टर (IR) मध्ये मेमोरी युनिट चे इन्स्ट्रक्शन आणलेले असतात. आय (I) बिट, ऑपरेशन कोड आणि बिट्स 0-11 हे घटक आहेत जे इन्स्ट्रक्शन रजिस्टर तयार करतात. 3x8 डीकोडर ऑपरेशन कोड बिट्स 12-14 एन्कोड करते. डीकोडरमध्ये D0 ते D7 असे लेबल केलेले आउटपुट आहेत. बिट 15 ऑपरेशन कोड I-फ्लिप-फ्लॉपमध्ये वाचला जातो. कंट्रोल लॉजिक गेट्स ऑपरेशन कोड 0-11 वापरतात. बायनरी सिकवेन्स काउंटर (SC) 0-15 मोजतो.

इन्स्ट्रक्शन अंमलात आणण्यासाठी प्रोसेसरला योग्य क्रमाने कंट्रोल सिग्नल तयार करावे लागतात. हार्डवायर्ड कंट्रोल डिजाइन साठी, AND गेट्स, OR गेट्स, एन्कोडर आणि डीकोडर यासारखे हार्डवेअर घटक वापरले जातात. कंट्रोल स्टेप ट्रॅक ठेवण्यासाठी काउंटरचा वापर केला जातो. हार्डवायर्ड कंट्रोल युनिटसाठी इनपुट म्हणजे क्लॉक, इन्स्ट्रक्शन रजिस्टर, कंडिशन कोड आणि डब्ल्यूएमएफसी (WFMC) आणि इंटरप्ट सारखे बाह्य इनपुट असतात.

प्रत्येक स्टेप ला एक क्लॉक सायकल लागते. स्टेप डिकोडरमध्ये प्रत्येक कंट्रोल सिकवेन्स टाइम युनिटसाठी सिग्नल लाईन्स असतात. इन्स्ट्रक्शन रजिस्टर (IR) चे ओपकोड बिट्स डीकोडरसाठी इनपुट असतात. इन्स्ट्रक्शन डीकोडर प्रत्येक मशीन इन्स्ट्रक्शनसाठी वन लाइन आउटपुट करते. इन्स्ट्रक्शन रजिस्टर मध्ये लोड केलेल्या कोणत्याही इन्स्ट्रक्शन साठी, IR [0-11] पासून एक आउटपुट लाइन 1 वर सेट केली आहे आणि इतर सर्व लाइन 0 वर सेट

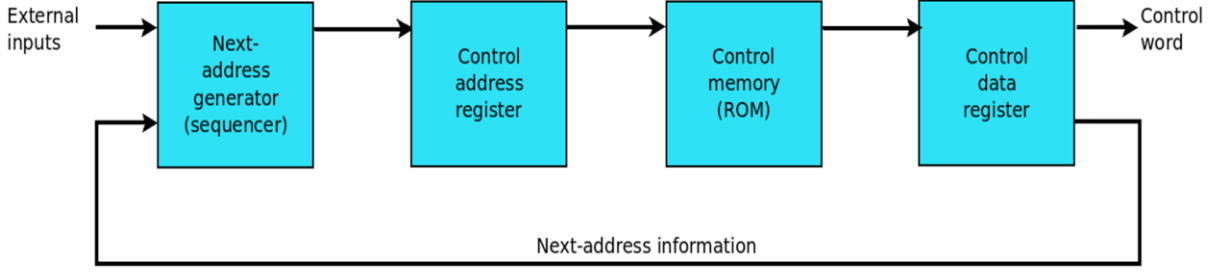
केल्या आहेत. (not active). एन्कोडर सर्व इनपुट सिग्नल एकत्र करतो आणि स्वतंत्र कंट्रोल सिग्नल तयार करतो. हार्डवायर्ड कंट्रोल युनिट मर्यादित इंस्ट्रक्शनसाठी कंट्रोल सिग्नल तयार करते.



आकृती. 2.3: हार्डवायर्ड कंट्रोल युनिट

2.3.2 मायक्रोप्रोग्राम कंट्रोल युनिट

मायक्रोप्रोग्राम केलेले कंट्रोल युनिट इंस्ट्रक्शन एक्सिक्यूशन साठी कंट्रोल सिग्नल्स तयार करतात. याचा वापर कॉम्प्लेक्स इंस्ट्रक्शन सेट कॉम्प्युटर (CISC) स्टाइल प्रोसेसर डिझाइन करण्यासाठी केला जातो. कंट्रोल सिग्नल निर्माण करणारा प्रोग्राम 'मायक्रोप्रोग्राम' म्हणून ओळखला जातो. हे मायक्रोप्रोग्राम विशिष्ट कंट्रोल मेमोरी मध्ये साठवले जातात.



आकृती. 2.4: मायक्रोप्रोग्रॅम कंट्रोल ऑर्गनायझेशन

आकृती 2.4 मध्ये मायक्रोप्रोग्रॅम कंट्रोल युनिट चे कार्य दर्शविते. रॉम (ROM) मेमोरी कंट्रोल डेटा कायमस्वरूपी होल्ड करून ठेवतात. कंट्रोल डेटा रजिस्टरमध्ये मेमोरी-रीड मायक्रोइंस्ट्रक्शन असते, तर कंट्रोल मेमोरी अॅड्रेस रजिस्टर त्याचा अॅड्रेस निर्दिष्ट करते.

मायक्रोइंस्ट्रक्शनमधील कंट्रोल वर्ड एक किंवा अधिक मायक्रोऑपरेशन्स परिभाषित करतो. ही टास्क पूर्ण झाल्यावर कंट्रोल पुढील अॅड्रेस निवडेल. पुढील मायक्रोइंस्ट्रक्शन अनुक्रमात किंवा कंट्रोल मेमोरी मध्ये असू शकते. परिणामी, सध्याच्या मायक्रोइंस्ट्रक्शन चा काही पार्ट पुढच्या अॅड्रेस वर परिणाम करतात. एक्सटर्नल इनपुट पुढील अॅड्रेस बदलू शकतात. पुढील अॅड्रेस जनरेटर सर्किट गणना करते आणि मायक्रोऑपरेशन्स दरम्यान पुढील मायक्रोइंस्ट्रक्शन वाचण्यासाठी कंट्रोल अॅड्रेस रजिस्टर पाठवते.

मायक्रोइंस्ट्रक्शन्स मायक्रोऑपरेशन्स सुरू करतात आणि कंट्रोल मेमोरी अॅड्रेस सिक्वेन्स सेट करतात. पुढील अॅड्रेस जनरेटर कंट्रोल मेमोरीमधून प्राप्त होणारा अॅड्रेस सिक्वेन्सर ठरवीतो आणि त्याला अनेकदा मायक्रोप्रोग्रॅम सिक्वेन्सर असे म्हणतात. मायक्रोप्रोग्रॅम सिक्वेन्सर कंट्रोल अॅड्रेस रजिस्टर मध्ये एकाने वाढ करतात. कंट्रोल मेमोरी तील अॅड्रेस, एक्सटर्नल अॅड्रेस ने किंवा प्रारंभिक अॅड्रेस ने कंट्रोल ऑपरेशन चे कार्य सुरू होतात.

पुढील अॅड्रेस कॉम्प्यूट करे पर्यंत आणि मेमोरी तून वाचला जाईपर्यंत कंट्रोल डेटा रजिस्टर सध्या एँक्टिव्ह मायक्रोइंस्ट्रक्शन स्टोर केल्या जातात. एक कॉम्बिनेशनल रॉम सर्किट अॅड्रेस ची व्हॅल्यू रीड करतात आणि वर्ड (word) चे आउटपुट देते.

जोपर्यंत अॅड्रेस चे व्हॅल्यू अॅड्रेस रजिस्टर मध्ये आहे तोपर्यंत आउटपुट वायरमध्ये रॉम (ROM) वर्डची सामग्री असते. रँडम ऍक्सेस मेमोरी (RAM) प्रमाणे, रीड सिग्नल ची आवश्यकता नाही. प्रत्येक क्लॉक पल्स कंट्रोल वर्ड मायक्रोऑपरेशन्स ला कार्यान्वित करते आणि कंट्रोल अॅड्रेस रजिस्टरमध्ये एक नवीन अॅड्रेस ट्रान्सफर करते. मायक्रोप्रोग्रॅम केलेले कंट्रोल युनिट, कंट्रोल मेमोरी मध्ये राहणारा वेगळा मायक्रोप्रोग्रॅम निर्दिष्ट करण्यास अनुमती देतात.

• **हॉरीझॉन्टल मायक्रोप्रोग्रॅम कंट्रोल युनिट:** हॉरीझॉन्टल मायक्रोप्रोग्रॅम कंट्रोल युनिट मध्ये

1. कंट्रोल वर्डमध्ये n बिट्स असतात, सर्व n बिट्स n कंट्रोल सिग्नल तयार करतात.
2. बाह्य इनपुटसाठी काही बिट्स, कंडिशनल जम्प सारख्या फंक्शनल कोडसाठी काही बिट्स, कंडिशन कोडसाठी काही बिट्स (फ्लॅग) आणि पुढील इंस्ट्रक्शन अॅड्रेस साठी काही बिट्स असतात.
3. या कंट्रोल युनिटमध्ये, कंट्रोल वर्डमध्ये बिट्सची संख्या जास्त असते (longer).

- **व्हर्टिकल मायक्रोप्रोग्रॅम कंट्रोल युनिट:** व्हर्टिकल मायक्रोप्रोग्रॅम कंट्रोल युनिट मध्ये

1. n बिट्स कंट्रोल शब्दांसाठी, केवळ $\log n$ कंट्रोल संकेत तयार केले जातात. समजा फंक्शनल कोडसाठी 64 बिट आहेत, सर्व 64 बिट फंक्शनल कोडसाठी फक्त 6 कंट्रोल सिग्नल तयार केले जातील.

2. कंट्रोल वर्डमध्ये बिट्सची संख्या कमी असते (shorter).

दोन्ही प्रकारच्या कंट्रोल युनिटची तुलना केल्यावर, हार्डवायर्ड कंट्रोल युनिटपेक्षा मायक्रोप्रोग्रॅम कंट्रोल युनिटचे खालील फायदे/तोटे होऊ शकतात:

1. हार्डवायर्ड कंट्रोल युनिटच्या तुलनेत मायक्रोप्रोग्रॅम कंट्रोल युनिट कमी खर्चिक आहे कारण ते प्रोग्रॅमसह अंमलात आणले जातात.
2. हार्डवायर्ड कंट्रोल युनिट्सच्या तुलनेत मायक्रोप्रोग्रॅम कंट्रोल युनिटमध्ये बदल करणे सोपे आहे. जर इंस्ट्रक्शन बदलले तर मायक्रोप्रोग्रॅम देखील सुधारित केले जातात परंतु हार्डवायर्ड कंट्रोल युनिटमध्ये बदल करणे कठीण आहे, कारण त्यात अरे लॉजिक गेट्स (AND, OR) आणि भरपूर वायरिंग असतात.
3. हार्डवायर्ड कंट्रोलच्या तुलनेत मायक्रोप्रोग्रॅम केलेले कंट्रोल युनिट हळू आहे कारण ते अनेक इंस्ट्रक्शनसाठी कंट्रोल सिग्नल तयार करतात.
4. हार्डवायर्ड कंट्रोल आरआयएससी (RISC) आर्किटेक्चरद्वारे वापरले जाते आणि मायक्रोप्रोग्रॅम केलेले कंट्रोल युनिट सीआयएससी (CISC) आर्किटेक्चरद्वारे वापरले जातात.

2.4 संगणक अरीथमेटिक

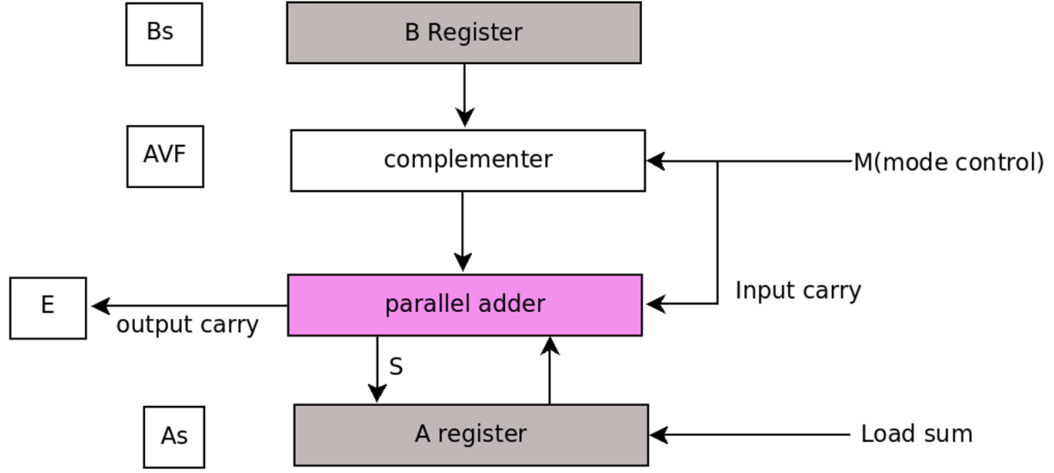
अरीथमेटिक ऑपरेशन्स, म्हणजे, बेरीज, वजाबाकी, गुणाकार आणि भागाकार हे साइन आणि अन-साइन दोन्ही पूर्णांकांवर केले जाऊ शकते. साइन आणि माग्रीटूड आणि $2s$ कॉम्पलिमेन्ट संख्यांच्या कार्यासाठी अरीथमेटिक हार्डवेअर कार्याची अंमलबजावणी तंत्रे या विभागात स्पष्ट केली आहेत.

2.4.1 साइन आणि माग्रीटूड संख्यांसाठी बेरीज आणि वजाबाकी

साइन आणि माग्रीटूड संख्या रजिस्टर A आणि B मध्ये संग्रहित केल्या जातात. फ्लिप-फ्लॉप As आणि Bs साइन बिट चे व्हॅल्यू धारण करतात. एकतर As किंवा Bs फ्लिप-फ्लॉपमध्ये फक्त एक बिट माहिती 0 किंवा 1 असते. एडिशन ओव्हरफ्लो रजिस्टर (AVF) मध्ये एक बिट माहिती असते आणि त्याचे व्हॅल्यू जोडणीनंतर ओव्हरफ्लो झाला की नाही हे दर्शवते. जर जोडणीचा परिणाम रजिस्टर साठवण क्षमतेपेक्षा जास्त असेल तर ओव्हरफ्लो होतो. ही ओव्हरफ्लो माहिती एव्हीएफ (AVF) मध्ये साठवली जातात.

बेरीज ऑपरेशनसाठी समांतर बेरीज सर्किट वापरून रजिस्टर B आणि A ची व्हॅल्यू जोडली जातात. जेव्हा M (मोड कंट्रोल) बिट जोडणीसाठी 0 वर सेट केले जाते, तेव्हा कॉम्पलिमेन्टर सर्किट कॉम्पलिमेन्ट न करता B रजिस्टरचे व्हॅल्यू प्रसारित करते. A आणि B रजिस्टरची व्हॅल्यू जोडण्याचा परिणाम रजिस्टर A आणि B मध्ये साठवला जातो. वजाबाकी ऑपरेशनसाठी, M (मोड कंट्रोल) बिट व्हॅल्यू 1 वर सेट करणे हे कॉम्पलिमेन्टर सर्किट B रजिस्टर व्हॅल्यूला कॉम्पलिमेन्ट करते आणि समांतर बेरीज सर्किट मध्ये ट्रान्सफर करते. समांतर बेरीज सर्किट B

रजिस्टरचे व्हॅल्यू, A रजिस्टरचे व्हॅल्यू आणि इनपुट कॅरी जोडते आणि आकृती 2.5 मध्ये दर्शविल्या प्रमाणे परिणाम A आणि As रजिस्टरमध्ये संग्रहित करतात.



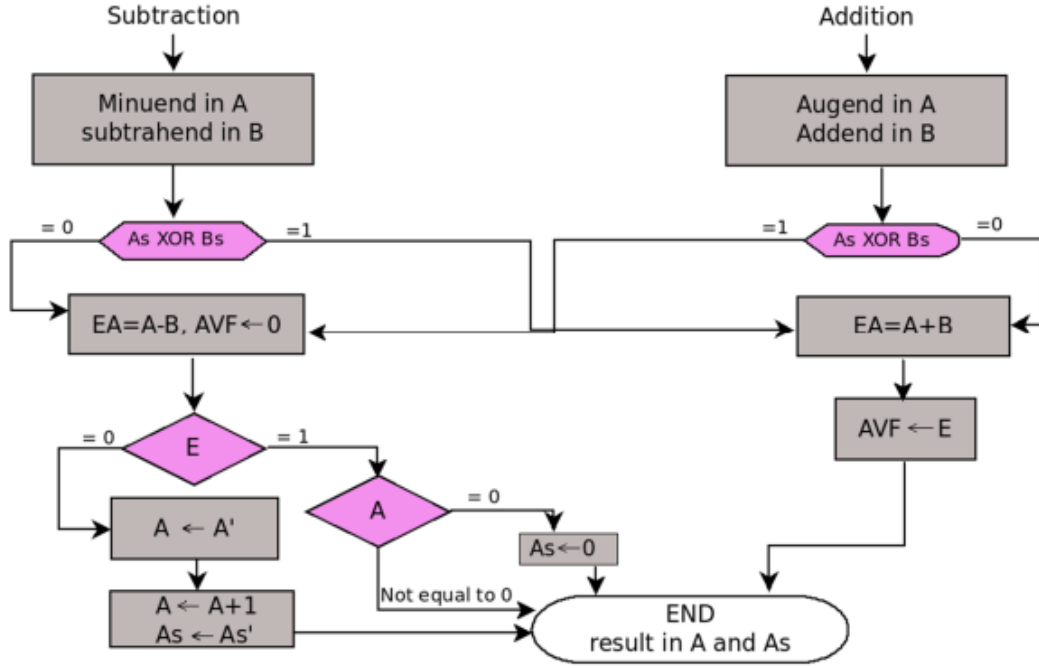
आकृती. 2.5: साइन आणि माग्नीट्यूड संख्या साठी बेरीज आणि वजाबाकी ऑपरेशन ची हार्डवेअर अंमलबजावणी

फ्लोचार्टचा वापर करून, आकृती. 2.6 मध्ये दर्शविल्या प्रमाणे साइन आणि माग्नीट्यूड संख्यांसाठी वजाबाकी आणि बेरीज चे कार्य स्पष्ट करतात. वजाबाकीसाठी, A आणि B ही रजिस्टर व्हॅल्यू A मध्ये मायन्यूएंड आणि B मध्ये सबस्ट्रॅन्ड सेटयित करतात. याव्यतिरिक्त, रजिस्टर A मध्ये अजेंड व्हॅल्यू आणि रजिस्टर B मध्ये अड्डेन्ड व्हॅल्यू संग्रहित केली जातात. बेरीज किंवा वजाबाकी करण्यापूर्वी, साइन बिट्सची तुलना एक्सओआर (XOR) ऑपरेशन वापरून केली जाते आणि जर एक्सओर (XOR) निकाल 1 असेल तर ते सूचित करते की साइन वेगळी आहेत. जर XOR चे आउटपुट 0 असेल तर दोन्ही साइन समान असतील.

जर XOR आउटपुट 0 असेल तर एडिशन ऑपरेशनचा परिणाम रजिस्टर EA मध्ये सेव्ह केला जातो. या ऑपरेशन मध्ये $EA \rightarrow A + B$, आउटपुट कॅरी व्हॅल्यू रजिस्टर E मध्ये साठवली जाते आणि परिणाम रजिस्टर A मध्ये संग्रहित केला जातो. जर एडिशन ऑपरेशननंतर ओव्हरफ्लो झाला तर तो एव्हीएफ (AVF) मध्ये सेव्ह केला जातो. जर चिन्हे भिन्न असतील (एक्सओर आउटपुट 1 असेल) तर वजाबाकी ऑपरेशन ($EA = A - B$) केले जाते. परिणाम A रजिस्टरमध्ये सेव्ह केला जातो, तर आउटपुट कॅरी रजिस्टर E मध्ये सेव्ह केली जाते. वजाबाकी प्रोसीजरत, ओव्हरफ्लो होणार नाही, एव्हीएफ (AVF) शून्यावर सेट केला आहे.

वजाबाकी करण्यापूर्वी, परिमाणांच्या साइन बिट्सची तुलना एक्सओआर (XOR) ऑपरेशन वापरून केली जाते. जर XOR आउटपुट 1 असेल, तर चिन्हे भिन्न आहेत, ऑपरेशन वजाबाकी आहे आणि दोन परिमाण जोडले जातात. जर XOR आउटपुट शून्य असेल, तर ते सूचित करते की दोन्ही परिमाण ची साइन समान आहेत आणि ऑपरेशन वजाबाकी आहे. सबट्रॅक्शन ऑपरेशनचे आउटपुट रजिस्टर E आणि A मध्ये साठवले जाते. आउटपुट 0 आणि 1 च्या तुलनेत E वाहून नेतात. जर E व्हॅल्यू 1 असेल म्हणजे $A > B$, तर ते सूचित करते की परिमाण A हे

B च्या परिमाणापेक्षा मोठे किंवा समान आहे. A च्या (परिणाम) व्हॅल्यूची तुलना 0 व्हॅल्यूशी केली जाते, जर समान असेल तर परिमाण A हे परिमाण B च्या बरोबरीचे आहेत.

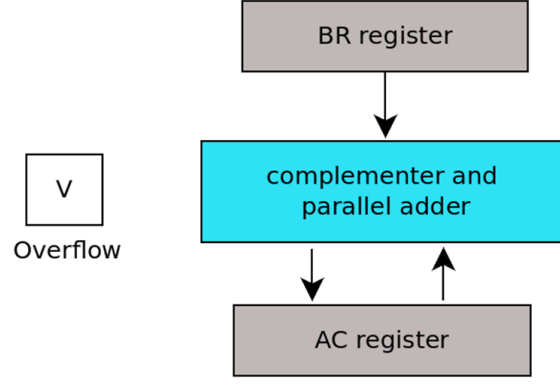


आकृती. 2.6: साइन आणि माय्नीटूड बेरीज आणि वजाबाकी ऑपरेशन चा फ्लोचार्ट

As चिन्ह 0 वर सेट केले आहे आणि परिणाम A आणि As मध्ये सेव्ह केला आहे. जर E हे 0 असेल, तर परिमाण A हे परिमाण B पेक्षा कमी असेल, तर परिणाम म्हणजे 2 चे कॉम्प्लिमेंट A रजिस्टरमध्ये सेव्ह केले जातात आणि As कॉम्प्लिमेंट साइन बिट 1 चे As फ्लिप-फ्लॉप मध्ये सेव्ह केले जातात.

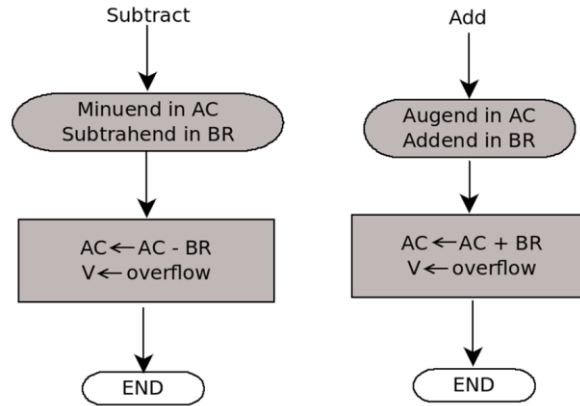
2.4.2 2 कॉम्प्लिमेंट संख्यांची बेरीज व वजाबाकी

पूर्णांकांचे चिन्ह बिट्स 2 कॉम्प्लिमेंट बेरीज आणि वजाबाकी ऑपरेशनमध्ये वेगळे केले जात नाहीत. साइन बिट्ससह क्रमांक AC (एक्युमुलेटर रजिस्टर) आणि BR (बेस रजिस्टर)मध्ये सेव्ह केले जातात. कॉम्पलीमेंटॉर आणि पॅरलल ऍंडर सर्किट मध्ये ओव्हरफ्लो होऊ शकतात. जेव्हा ओव्हरफ्लो होतो, तेव्हा आकृतीमध्ये 2.7 मध्ये दर्शविल्याप्रमाणे ओव्हरफ्लो स्टेटस रजिस्टर V हा 1 वर सेट केला जातो. आउटपुट कॅरी टाकून दिली जातात.



आकृती. 2.7: 2s कॉम्प्लिमेंट संख्यांसाठी बेरीज आणि वजाबाकी च्या हार्डवेअर ची अंमलबजावणी

2s कॉम्प्लिमेंट संख्यांच्या बायनरी पूर्णांकांची बेरीज आणि वजाबाकी करण्याची पद्धत आकृती 2.8 मध्ये दर्शविली आहे. साइन बिट्स आणि बेस रजिस्टर (BR) बायनरी व्हॅल्यू असलेले एक्ज्युटिव्ह रजिस्टर (AC) बायनरी व्हॅल्यू जोडले जातात. अंतिम दोन बिट्सच्या एक्सक्लुझिव्ह OR मध्ये 1 असल्यास, ओव्हरफ्लो V 1 वर सेट केला जातो; अन्यथा, तो 0 वर सेट केला जातो. वजाबाकी प्रोसीजरदरम्यान बी. आर. रजिस्टर व्हॅल्यूच्या 2-बिट कॉम्प्लिमेंट मध्ये एसी रजिस्टर व्हॅल्यू जोडले जाते आणि उलटपक्षी.



आकृती. 2.8: 2 कॉम्प्लिमेंट संख्यांसाठी बेरीज आणि वजाबाकी चा फ्लोचार्ट

2.4.3 पूर्णांक गुणाकार

साइन पूर्णांकांचा गुणाकार करण्यासाठी विविध तंत्रे आहेत. या विभागात, साइन आणि माग्नीट्यूड साठीच्या विविध गुणाकार पद्धती आणि 2s कॉम्प्लिमेंट संख्या प्रणालीवर चर्चा केली आहेत.

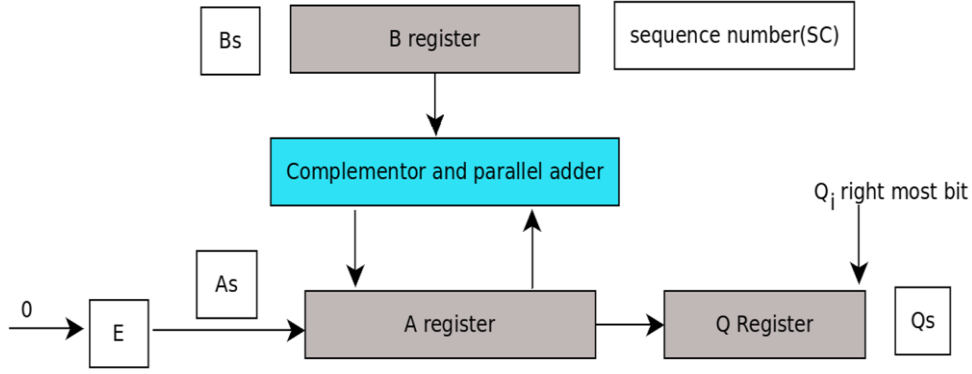
1. साइन आणि माग्नीट्यूड संख्यांचा गुणाकार:

साइन आणि माग्नीट्यूड गुणाकार ऑपरेशनसाठी, हार्डवेअर घटक म्हणजे रजिस्टर A, B आणि Q, सिक्वेन्स काउंटर रजिस्टर (एससी) कॉम्प्लिमेण्टर आणि पॅरलल ऍंडर सर्किट आणि As, Qs, Bs, Qi फ्लिप-फ्लॉप.



स्कॅन करा

उदाहरणार्थ, चिन्ह आणि परिमाण संख्यांचा गुणाकार



आकृती. 2.9: साइन आणि माग्नीट्यूड संख्या गुणाकार

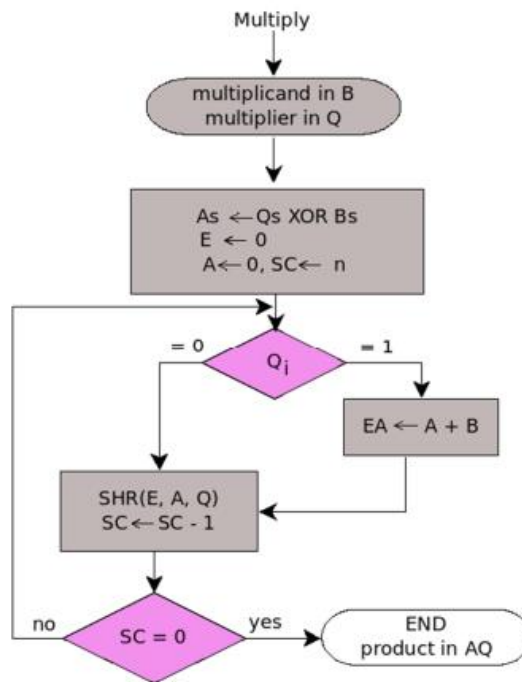
दोन साइन आणि माग्नीट्यूड पूर्णांकांचा गुणाकार करण्यासाठी आवश्यक असणारे रजिस्टर्स, हार्डवेयर आणि स्टेप आकृती 2.9 मध्ये दर्शविल्या गेलेले आहेत. मल्टीप्लिकेन्ड रजिस्टर A मध्ये साठवले जाते, तर मल्टीप्लायर रजिस्टर Q मध्ये साठवले जाते. रजिस्टर A चे सुरुवातीचे व्हॅल्यू 0 आहे. A आणि B रजिस्टरच्या व्हॅल्यूचा गुणाकार केल्यानंतर, आंशिक आउटपुट रजिस्टर A मध्ये साठवले जाते. सिक्वेन्स काउंटर n वर सेट केले आहे (s वगळता गुणकातील बिट्सची संख्या).

आकृती 2.10 मध्ये रजिस्टर A आणि E ची सुरुवात '0' अशी केली आहे. XOR ऑपरेशन मल्टिप्लिकँड साइन बिट आणि मल्टिप्लायर साइन बिट वापरून कार्यान्वित केले जातात, ज्याचा परिणाम बिट As फ्लिप-फ्लॉपमध्ये संग्रहित केला जातो. जर रजिस्टर Q चा सर्वात उजवा बिट $Q_i = 0$ असेल, तर E, A आणि Q वर शिफ्ट-राईट ऑपरेशन केले जाते आणि SC काउंटर एकाने कमी केला जातो. $Q_i = 1$ असल्यास, रजिस्टर A आणि B ची व्हॅल्यू पॅरलल ऍंडर सर्किट वापरून जोडली जातात आणि आंशिक आउटपुट रजिस्टर A मध्ये साठवले जातात. ही प्रक्रिया सिक्वेन्स काउंटर (एससी) 0 पर्यंत पोहोचेपर्यंत सुरू ठेवली जाते आणि परिणाम रजिस्टर A, Q मध्ये साठवला जातो.

उदाहरण 2.1: साइन आणि माग्नीट्यूड पद्धत वापरून -6 आणि 3 च्या गुणाकारांची गणना करा.

उपाय: मल्टीप्लिकेन्ड ही संख्या -6 आहे, तर मल्टीप्लायर ही संख्या 3 आहे. मल्टीप्लिकेन्ड -6 हे रजिस्टर B मध्ये खालील प्रकारे साठवले जातात. फ्लिप-फ्लॉप Bs चे व्हॅल्यू 1 आहे, कारण गुणाकार नकारात्मक आहे आणि परिमाण 0110 हे B मध्ये साठवले आहे.

मल्टीप्लायर 3 हा 0011 या बायनरी व्हॅल्यूसह Q रजिस्टरमध्ये साठवला जातो आणि Qs हा 0 वर सेट केला जातो. As फ्लिप-फ्लॉप मध्ये प्रॉडक्ट आउटपुट चे चिन्ह असते; जेव्हा XOR ऑपरेशन Qs (म्हणजे, 0) आणि Bs (म्हणजे, 1) वर सेट केले जाते तेव्हा As हे 1 वर सेट केले जाते. रजिस्टर E आणि A, 0 ने प्रारंभ केले आहेत आणि SC एकूण बिट्सच्या संख्येसह प्रारंभ केले आहे, म्हणजेच 4, मल्टीप्लायर मध्ये. अंतिम परिणाम A आणि Q मध्ये संग्रहित केला जातो; $AQ = 00010010$, किंवा 18 दशांश मध्ये. As हे 1 असल्याने, अंतिम गुणाकार-18 ($AQ = 10010010$) आहे. तक्ता 2.1 मध्ये गुणाकारासाठी कॉम्प्युटेशन आणि इंटरमीडिएट रजिस्टर व्हॅल्यू दर्शविली आहेत.



आकृती. 2.10: साइन आणि माग्नीट्यूड गुणाकार फ्लोचार्ट

तक्ता 2.1: साइन आणि माग्नीट्यूड संख्या गुणाकार उदाहरण (-6x3)

Multiplicand in B=0110	E	A	Q	SC
Multiplier in Q	0	0000	0011	4
$Q_i = 1$; add B to A		0110		
First partial product in A	0	0110	0011	
Shift right(EAQ)	0	0011	0001	3
$Q_i=1$; add B to partial product A		0110		
Partial product in A	0	1001	0001	
Shift right (EAQ)	0	0100	1000	2
$Q_i=0$; shift right(EAQ)	0	0010	0100	1
$Q_i=0$; shift right(EAQ)	0	0001	0010	0
$A_s=1$; set $A_s=1$ in A for final product (AQ)	0	1001	0010	

2. 2s कॉम्पेलमेंट संख्यांसाठी बूथचा गुणाकार

गुणाकार करण्यासाठी बूथच्या दृष्टिकोनात, AC (एकमूलॅटोर) BR (बेस रजिस्टर) QR (भागफल) SC (सिक्वेन्स काउंटर) आणि कॉम्पलीमेंटॉर आणि पॅरलल ऍडर सर्किट वापरले जातात.

फ्लिप-फ्लॉप Q_i हा मल्टीप्लेयर सर्वात उजवीकडील बिट आहे, तर Q_{i-1} हा फ्लिप-फ्लॉप मधील आहेत. या मल्टिप्लिकेशन ऑपरेशनमध्ये, मल्टिप्लिकेन्ड रजिस्टर बीआर (BR) मध्ये संग्रहित केला जातो, मल्टिप्लायर क्यूआर (QR) रजिस्टरमध्ये संग्रहित केला जातो, AC रजिस्टरचे प्रारंभिक व्हॅल्यू शून्य असते आणि AC व्हॅल्यू हे साइन बिटसह मल्टिप्लायरमधील बिट्सची संख्या असते.

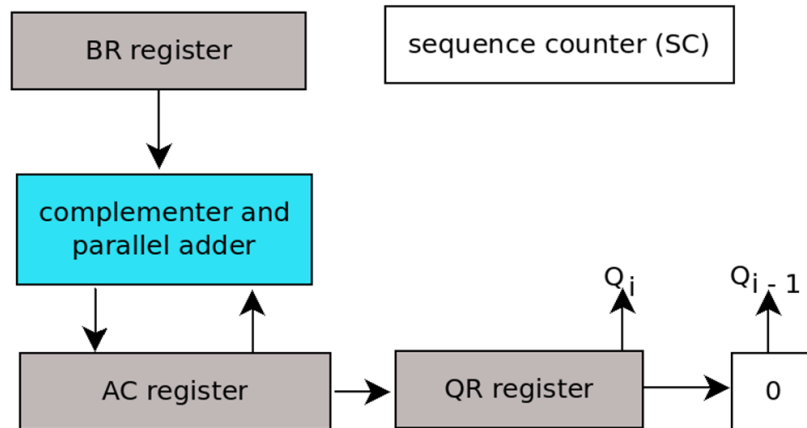
बूथच्या गुणाकारची हार्डवेअर अंमलबजावणी आकृती 2.11. मध्ये दर्शविली आहे. Q_{i-1} फ्लिप-फ्लॉपचे प्रारंभिक व्हॅल्यू शून्य आहे आणि क्यूई हे गुणकातील सर्वात उजवीकडील बिट आहे. Q_{i-1} च्या व्हॅल्यूवर अवलंबून, एकतर बेरीज किंवा वजाबाकी केली जाते. जर $Q_i Q_{i-1}$ हे 00 किंवा 11 असेल, तर AC, QR आणि Q_{i-1} च्या बायनरी व्हॅल्यूवर अरीथमेटिक राइट शिफ्ट केले जाते आणि SC हे एकाने कमी केले जाते. जेव्हा $Q_i Q_{i-1}$ हे 01 च्या बरोबरीचे असते, तेव्हा बेरीज केली जातात.

AC, QR, Q_{i-1} आणि SC वरील अरीथमेटिक राइट शिफ्ट जोडणीनंतर एकाने कमी केले जाते. जर $Q_i Q_{i-1}$ हे 10 च्या बरोबरीचे असेल तर वजा केले जातात. AC, QR, Q_{i-1} आणि SC वरील योग्य शिफ्ट वजा केल्यानंतर एकाने कमी होते. ज्या प्रकारे आकृती 2.12 मध्ये दर्शविल्या गेलेले आहेत, सिक्वेन्स काउंटर (एससी) 0 पर्यंत पोहोचेपर्यंत या प्रक्रिया सुरू ठेवल्या जातात.

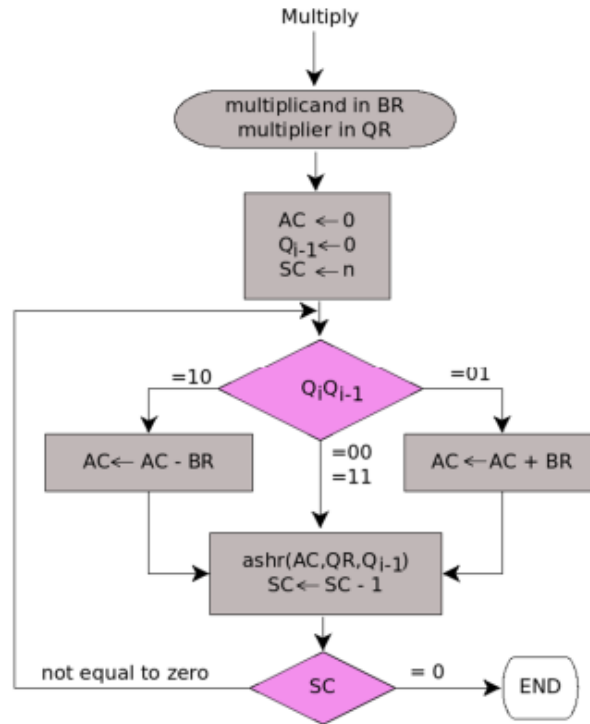


स्कॅन करा

अधिक उदाहरणासाठी
बूथच्या गुणाकारावर



आकृती. 2.11 बूथ गुणन यंत्रसामग्री कार्यान्वयन



आकृती. 2.12: बूथचा गुणाकार फ्लोचार्ट

उदाहरण 2.2 बूथ गुणाकार पद्धतीचा वापर करून तुम्ही -5 आणि -4 संख्यांचा गुणाकार कसा कराल?

उपाय: रजिस्टर BR मध्ये गुणाकार-5 चे 2s कॉम्प्लिमेन्ट असते, जे 1011 आहे. गुणक -4 चे 2s चे कॉम्प्लिमेन्ट क 1100 आहे, जे QR रजिस्टरमध्ये साठवले जाते. AC रजिस्टरचे प्रारंभिक व्हॅलू 0000 आहे आणि गुणकाला चार बिट्स असल्याने SC हे 4 वर सेट केले आहेत.

तक्ता 2.2: बूथ गुणनाचे उदाहरण (-5 x -4)

$Q_i Q_{i-1}$	BR=1011 2's comp of BR = 0101	AC	QR	Q_{i-1}	SC
	Initial	0000	1100	0	4
0 0	ashr(AC, QR, Q_{i-1})	0000	0110	0	3
0 0	ashr(AC, QR, Q_{i-1})	0000	0011	0	2
1 0	subtract BR	0101			
		0101			
	ashr(AC, QR, Q_{i-1})	0010	1001	1	1
1 1	ashr(AC, QR, Q_{i-1})	0001	0100	1	0

बेरीज किंवा वजाबाकी ऑपरेशन करणे, $Q_i Q_{i-1}$ च्या व्हॅल्यूवर अवलंबून असते. त्यानंतर अरीथमेटिक राइट शिफ्ट केले जाते. या परिस्थितीत, जर $Q_i Q_{i-1}$ हे 00 किंवा 11 असेल तर अरीथमेटिक शिफ्ट केले जाते. AC चे

व्हॅलू 0 पर्यंत पोहोचेपर्यंत ही प्रक्रिया सुरू ठेवली जाते. अंतिम आउटपुट साठवण्यासाठी AC आणि QR रजिस्टरचा वापर केला जातो. टेबल 2.2 मध्ये, प्रत्येक टप्प्यासाठी रजिस्टर आणि फ्लिप-फ्लॉप व्हॅलू तपशीलवार दिली आहेत.

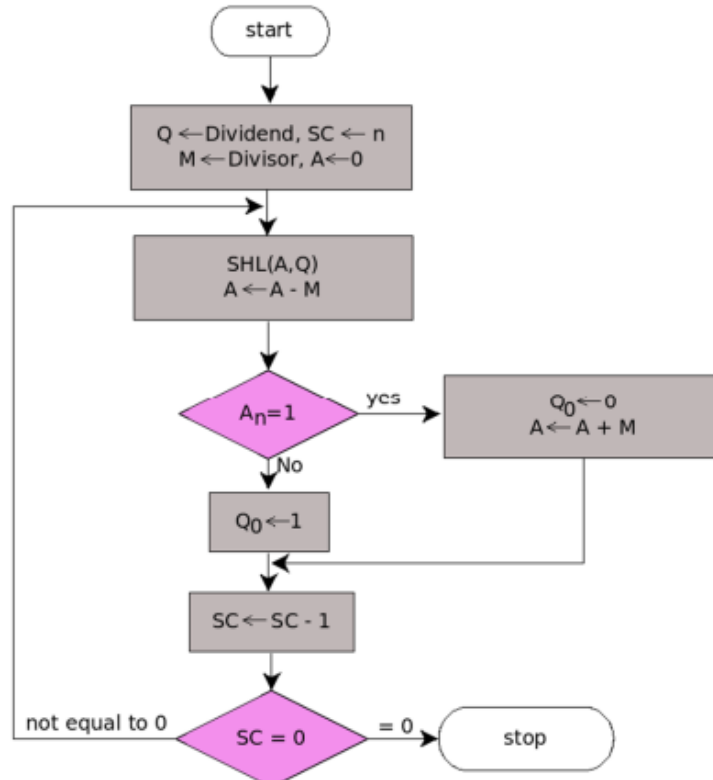
2.4.4 पूर्णांक भागाकार

भागाकार करण्यासाठी दोन पद्धती आहेत: रिस्टोरिंग भागाकार पद्धत आणि नॉन रिस्टोरिंग भागाकार पद्धत.

1. रिस्टोरिंग भागाकार:

प्रारंभिक व्हॅल्यू रजिस्टर A, Q, M आणि AC मध्ये स्टोर केली जातात. (sequence counter). रजिस्टर M हा n-बिट बायनरी व्हॅल्यूचा सकारात्मक विभाजक संचयित करतो, तर रजिस्टर Q हा सकारात्मक लाभांश सेटयित करतो. रजिस्टर A मध्ये $n + 1$ बिट्स आहेत, जे सर्व 0 वर सेट केलेले आहेत. सिक्वेन्स काउंटर 0 ने प्रारंभ केला आहे.

आकृती 2.13 रिस्टोरिंग भागाकार करण्यासाठी फ्लोचार्ट दर्शविते. (A, Q) रजिस्टर व्हॅल्यूवर अंमलात आणलेले पहिले ऑपरेशन शिफ्ट लेफ्ट आहे. त्यानंतर, रजिस्टर A च्या व्हॅल्यूतून रजिस्टर M चे व्हॅल्यू वजा केले जाते आणि परिणामी व्हॅल्यू रजिस्टर A मध्ये साठवले जातात. A_n द्वारे दर्शविलेल्या रजिस्टर A च्या साइन बिटची तुलना आता शून्याशी केली जाते. जर A_n हा 1 असेल, तर Q_0 हा 0 वर सेट केला जातो आणि A हा M जोडून रिस्टोरिंग केला जातो; अन्यथा, Q_0 हा 1 वर सेट केला जातो. हे टप्पे पूर्ण झाल्यावर AC चे व्हॅल्यू एकाने कमी केले जातात. AC व्हॅल्यू 0 पर्यंत पोहोचेपर्यंत प्रक्रिया सुरू ठेवली जाते.



आकृती 2.13: रिस्टोरिंग भागाकार फ्लोचार्ट

उदाहरण 2.3 रिस्टोरिंग भागाकार पद्धतीचा वापर करून 7/3 भागाकार करणे?

उपाय: रजिस्टर Q चे व्हॅल्यू 0111 (लाभांश) रजिस्टर A चे व्हॅल्यू 00000 आहे आणि रजिस्टर M चे विभाजक बायनरी व्हॅल्यू 00011 आहे. M चे 2 चे कॉम्प्लिमेंट 11101 आहे, जे सबट्रॅक्शन ऑपरेशन दरम्यान A रजिस्टर करण्यासाठी जोडले जाते. सिक्वेन्स काउंटरच्या स्टेप टेबल 2.3 मध्ये दर्शविल्या आहेत.

टेबल 2.3 मध्ये, 7/3 भागाकार ऑपरेशन नंतर रजिस्टर A चे व्हॅल्यू उर्वरित (0001) आहे आणि रजिस्टर Q चे व्हॅल्यू भागाकार आहे (0010). परिणामाची अचूकता 7/3 वर पूर्णांक विभागणी करून पुष्टी केली जाऊ शकते, ज्यामुळे भागफल व्हॅल्यू 2 आणि उर्वरित व्हॅल्यू 1 मिळतात.

2. नॉन रिस्टोरिंग भागाकार:

नॉन-रिस्टोरिंग डिव्हिजन तंत्र मध्यवर्ती टप्प्यांऐवजी डिव्हिजन ऑपरेशनच्या शेवटी आंशिक उर्वरित री-स्टोर करते. भागाकार प्रोसीजरसाठी A, M, Q आणि SC या रजिस्टर चा वापर करणे आवश्यक आहेत. जेव्हा भागाकार प्रोसेस पूर्ण होतात, तेव्हा रजिस्टर A चे व्हॅल्यू रेमेन्डर असते आणि रजिस्टर Q चे व्हॅल्यू भागाकार असते. रजिस्टर A चे प्रारंभिक व्हॅल्यू 0 आहे आणि विभाजकातील बिट्सची संख्या SC ला नियुक्त केली आहे.

तक्ता 2.3: रिस्टोरिंग भागाकार अल्गोरिदमचे

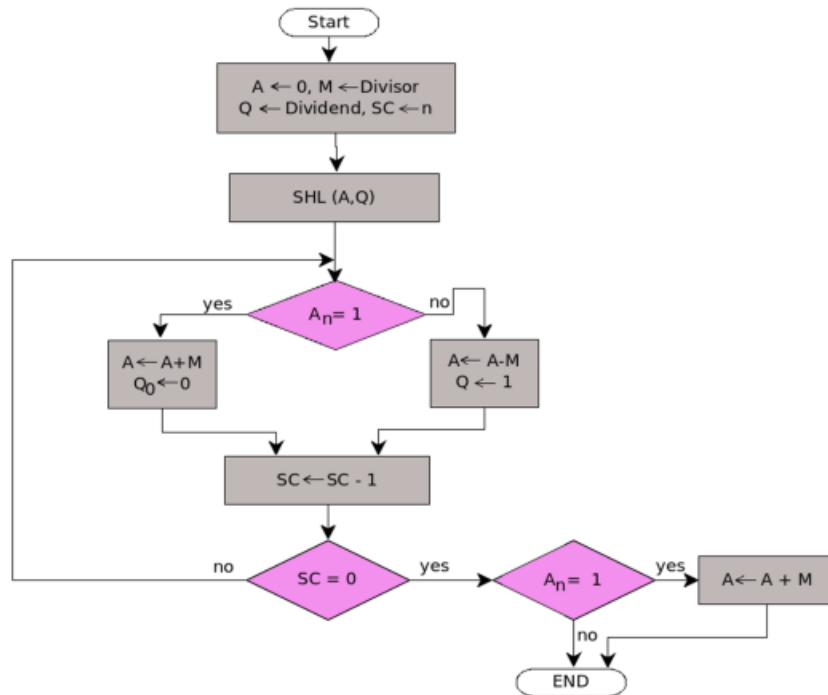
Steps	A	Q	SC
Initial	00000	0111	4
Shift left (A,Q)	00000	1110	
$A \leftarrow A - M$	11101		
$A_4=1$, so $Q_0 \leftarrow 0$	11101	1110	
M	00011		3
$A \leftarrow A + M$	00000	1110	
Shift left (A,Q)	00001	1100	
$A \leftarrow A - M$	11101		
$A_4=1$, so $Q_0 \leftarrow 0$	11110	1100	2
M	00011		
$A \leftarrow A + M$	00001	1100	
Shift left (A,Q)	00011	1000	
$A \leftarrow A - M$	11101		1
Carry is discarded	00000		
$A_4=0$; $Q_0 \leftarrow 1$	00000	1001	
Shift left (A,Q)	00001	0010	
$A \leftarrow A - M$	11101		0
$A_4=1$, so $Q_0 \leftarrow 0$	11110	0010	
M	00011		
$A \leftarrow A + M$	00001	0010	

आकृती. 2.14 मध्ये नॉन-रिस्टोरिंग डिव्हिजन तंत्राचा वापर करून भागाकार कार्यान्वित करण्यासाठी आवश्यक पावले दर्शविते. जर A_n हा 1 असेल, तर M ला A मध्ये जोडा आणि Q_0 ला 0 वर सेट करा; अन्यथा, A पासून M वजा करा आणि Q_0 ला 1 वर सेट करा. त्यानंतर सिक्वेन्स काउंटर एससी (SC) ची व्हॅल्यू एकाने कमी केले जाते. एससी (SC) व्हॅल्यू शून्यापर्यंत पोहोचेपर्यंत ही प्रक्रिया सुरू ठेवली जाते. जेव्हा भागाकार प्रक्रिया पूर्ण होते, तेव्हा रजिस्टर A चा साइन बिट तपासला जातो; जर A_n हा 1 असेल, तर रजिस्टर M ची व्हॅल्यू रजिस्टर A मध्ये जोडले जाते. (partial remainder added with divisor). अन्यथा, भागाकार प्रक्रिया संपुष्टात येते आणि रजिस्टर A ची व्हॅल्यू उर्वरित असते आणि रजिस्टर Q ची व्हॅल्यू भागाकार असते.

उदाहरण 2.4 नॉन-रिस्टोरिंग डिव्हिजन पद्धत वापरून तुम्ही $7/3$ भागाकार कशी करता?

उपाय: रजिस्टर चे व्हॅल्यू खालीलप्रमाणे निर्दिष्ट केले आहेत. रजिस्टर Q ला 0111 (डिविडेंड) असे सेट केले आहे तर रजिस्टर A ला 0 असे सेट केले आहे. कारण रजिस्टर A मध्ये $n + 1$ बिट्स आहेत, सर्व पाच बिट्स शून्य, i.e., 00000 वर सेट केले आहेत. रजिस्टर M चे व्हॅल्यू 00011 आहे. (डीवायजर). M चे 2 चे पूरक 11101 आहे. वजाबाकी प्रोसीजरदरम्यान, रजिस्टर M चे $2s$ कॉम्प्लीमेंट रजिस्टर A मध्ये साठवल्या जाते. तक्ता 2.4 मध्ये प्रत्येक रेजिस्टर व्हॅल्यूवर केलेल्या सिक्वेन्स काउंटरच्या स्टेप्स आणि कृती दर्शविल्या आहेत.

साइंड ऑपरेंडवर थेट भागाकार करण्यासाठी कोणतीही सरळ तंत्रे नाहीत. भागाकार मध्ये, प्रीप्रोसेसिंग वापरून ऑपरेंडचे सकारात्मक व्हॅल्यूमध्ये रूपांतर केले जाऊ शकते. साइंड संख्यांच्या भागाकार साठी, भाज्यक आणि शेष चिन्हे रिस्टोरिंग किंवा नॉन रिस्टोरिंग भागाकार करण्याच्या पद्धती वापरल्यानंतर सुधारित केली जाऊ शकतात.



आकृती. 2.14: नॉन रिस्टोरिंग भागाकार

तक्ता 2.4: नॉन रिस्टोरिंग भागाकार करण्याचे उदाहरण

Steps	A	Q	SC
Initial	00000	0111	4
Shift left (A,Q)	00000	1110	
Subtract M ($A \leftarrow A-M$)	11101		
	11101		
$A_4=1; Q_0 \leftarrow 0$	11101	1110	3
Shift left (A,Q)	11011	1100	
Add M ($A \leftarrow A+M$)	00011		
$A_4=1; Q_0 \leftarrow 0$	11110	1100	2
Shift left (A,Q)	11101	1000	
Add M ($A \leftarrow A+M$)	00011		
	00000		
$A_4=0; Q_0 \leftarrow 1$	00000	1001	1
Shift left (A,Q)	00001	0010	
Subtract M ($A \leftarrow A-M$)	11101		
$A_4=1; Q_0 \leftarrow 0$	11110	0010	0
If $SC=1$ and $A_4=1$; add M ($A \leftarrow A+M$)	00011		
	00001	0010	

2.5 अपूर्णांक संख्या प्रतिनिधित्व

अपूर्णांक संख्या फिक्स्ड पॉइंट प्रतिनिधित्व आणि फ्लोटिंग पॉइंट प्रतिनिधित्व या दोन स्वरूपात दर्शविल्या जातात.

2.5.1 फिक्स्ड पॉइंट प्रतिनिधित्व

दशांश पॉइंट फिक्स्ड-पॉइंट संकेतांकामध्ये हलत नाही. रूपांतरित बाइनरी संख्येतील दशांश पॉइंट मूळ दशांश व्हॅलुनुसार त्याच ठिकाणी स्थित असतो. उदाहरणार्थ, दिलेल्या दशांश क्रमांक 41.6875 चा एक फिक्स्ड दशांश पॉइंट आहे. अशा प्रकारे, दशांश पॉइंटपूर्वीचे अंक बाइनरी स्वरूपात दर्शविले जातात आणि दशांश पॉइंटनंतरचे अंक बाइनरी स्वरूपात रूपांतरित केले जातात. जर संपूर्ण संख्या बायनरीच्या 25 बिट्समध्ये दर्शविली गेली असेल तर 16 बिट्स दशांश पॉइंटच्या आधीची संख्या दर्शवतात, 8 बिट्स दशांश पॉइंटनंतरचे अंक दर्शवितात आणि 1 बिट साइन दर्शविते.

----- 25 bits -----		
1 बिट्स साइन करिता	16 बिट्स दशांश पॉइंटपूर्वी	8 बिट्स दशांश पॉइंटनंतर

2.5.2 फ्लोटिंग पॉइंट प्रतिनिधित्व

फ्लोटिंग पॉइंट रिप्रेझेंटेशनमध्ये, दशांश पॉइंट एकतर डावीकडे किंवा उजवीकडे शिफ्ट केला जातो. दशांश पॉइंटच्या हालचालीवर आधारित, घातांक (एक्सपोनंट) एकतर वाढवला किंवा कमी केला जातो. फिक्स्ड पॉइंट रिप्रेझेंटेशनमध्ये खूप मोठ्या किंवा लहान अपूर्णांक संख्यांचे प्रतिनिधित्व करू शकत नाही, जी एक कमतरता आहे. फ्लोटिंग पॉइंट रिप्रेझेंटेशनचा वापर करून, तथापि, लहान आणि मोठ्या दोन्ही संख्यांचे प्रतिनिधित्व केले जाऊ शकते. आयईईई 754 (IEEE 754) स्वरूप फ्लोटिंग पॉइंट नंबर्स दर्शविण्यासाठी वापरले जाते.

1. आयईईई 754 (IEEE 754) सिंगल प्रिसिजन रेप्रेसेंटेशन: सिंगल प्रिसिजन रिप्रेझेंटेशनमध्ये, फ्लोटिंग-पॉइंट व्हॅल्यू पॉझिटिव्ह किंवा निगेटिव्ह आहे की नाही हे पहिला बिट सूचित करतो. 8-बिट घातांकासह, अन साइनड पूर्णांकांची रेंज 0 ते 255 आहे, त्यात 0 आणि 255 विशिष्ट वापरासाठी राखीव आहेत. हे दोन पूर्णांक वगळता, बॅईस घातांकांची श्रेणी 1 आणि 254 च्या दरम्यान आहे.

-----32 बिट्स -----		
साइन	एक्सपोनंट	मॅटिसा
-----1 बिट्स -----	-----8 बिट्स -----	-----23 बिट्स -----

वास्तविक घातांक = बायस्ड घातांक- ऍक्सेस 127

सिंगल प्रिसिजन रिप्रेझेंटेशनसाठी अतिरिक्त 127 = (2^8-1) , ऑक्टल घातांक -126 (1-127) ते 127 पर्यंत असतो. (254-127). बायस घातांकाकडून वास्तविक घातांक मिळविण्यासाठी, अतिरिक्त 127 वजा केले जातात.

उदाहरण 2.5 सिंगल प्रिसिजन रिप्रेझेंटेशन मध्ये, आपण -14.25 कसे व्यक्त कराल?

उपाय: सिंगल प्रिसिजन रिप्रेझेंटेशन खालील प्रमाणे कॉम्प्युट करतात:

- 14.25 ची बायनरी व्हॅल्यू 1110.01 आहे आणि घातांक असलेली ही बायनरी संख्या 1110.01×2^0 आहे.
- सामान्यीकरण स्वरूप $1.M \times 2^{(घातांक-127)}$ दिले संख्या 1.11001×2^3 म्हणून सामान्यीकृत स्वरूपात दर्शविले जाऊ शकते आहे. येथे, वास्तविक घातांक 3 आहे, तथापि तो 8-बिट बायस्ड घातांक स्वरूपात व्यक्त केला जाईल.
- बायस्ड घातांक = (वास्तविक घातांक + ऍक्सेस 127)
- येथे, बायस्ड घातांक = $3 + 127 = 130$. तर बायस्ड घातांक 8 बिट बायनरी स्वरूपात 10000010 मध्ये संग्रहित केला आहे आणि साइन बिट 1 आहे.
- हणून, दिलेला क्रमांक -14.25 IEEE 754 सिंगल प्रिसिजन फॉरमाटमध्ये 110000010.111000000000000000 म्हणून 8 दर्शविला जातात.

2. आयईईई 754 (IEEE 754) डबल प्रिसिजन रिप्रेझेंटेशन:

आयईईई 754 डबल प्रिसिजन फ्लोटिंग पॉइंट रिप्रेझेंटेशन 64-बिट प्रणालींचा वापर करते. पहिला बिट साइन बिटचे प्रतिनिधित्व करतो, त्यानंतर 11 बिट्स घातांकाचे प्रतिनिधित्व करतात आणि 52 बिट्स मान्टिसाचे प्रतिनिधित्व करतात.

-----64 बिट्स -----		
साइन	एक्सपोनंट	मॅटिसा
-----1 बिट्स -----	-----11 बिट्स -----	-----52 बिट्स -----

वास्तविक घातांक = (बायस्ड घातांक- ऍक्सेस 1023)

येथे, ऍक्सेस 1023 = $2^{11-1}-1$. अन साइन घातांकांची रेंज 0 ते 2048 आहे, तथापि 0 आणि 2048 विशिष्ट हेतूसाठी राखीव आहेत. जर अतिरिक्त व्हॅल्यू 1023 वजा केले तर उर्वरित श्रेणी -1022 ते 1023 आहे.

2.5.3 फ्लोटिंग पॉइंट एरिथमेटिक ऑपरेशन्स

बेरीज, वजाबाकी, गुणाकार आणि भागाकार ही सर्वात सामान्य फ्लोटिंग पॉइंट ऑपरेशन्स आहेत. साधेपणासाठी, या क्रिया दशांश संख्यांवर दर्शविल्या जातात. तथापि, बायनरी संख्यांवर देखील समान प्रक्रिया केल्या जाऊ शकतात.

1. बेरीज किंवा वजाबाकी:

फ्लोटिंग-पॉइंट एरिथमेटिक बेरीज किंवा वजाबाकीसाठी, दोन्ही घातांक समतुल्य आहेत की नाही हे निर्धारित करणे आवश्यक आहे. समान नसल्यास, मान्टिसामधील दशांश पॉइंट अलाइन करण्यासाठी उजवीकडे किंवा डावीकडे हलविला जातो.

उदाहरणार्थ, खालील दोन फ्लोटिंग पॉइंट नंबर्सची बेरीज खालीलप्रमाणे केली जाऊ शकते.

$$0.5372400 \times 10^2$$

$$0.1580000 \times 10^{-1}$$

2 आणि -1 या दोन संख्यांचे घातांक समतुल्य नाहीत. पहिल्या/दुसऱ्या क्रमांकाची मान्टिसा उजवीकडे/डावीकडे हलवून ती संरेखित केली जाऊ शकते. दुसरी संख्या उजवीकडे तीन ऍड्रेसवर हलवली जाते. पहिली किंवा दुसरी संख्या मान्टिसा डावीकडे हलवल्यास, सर्वात महत्वाचे अंक नष्ट होतात. पहिल्या पूर्णांकावर उजवीकडे सरकत असताना, सर्वात कमी लक्षणीय बिट्स टाकून दिले जातात. या उदाहरणात, दुसरा क्रमांक तीन ऍड्रेसनी उजवीकडे सरकला आहे.

$$0.5372400 \times 10^2$$

$$+0.0001580 \times 10^2$$

$$0.5373980 \times 10^2$$

आकृती 2.15 मध्ये दर्शविल्याप्रमाणे रजिस्टर AC आणि BR मध्ये साठवलेल्या फ्लोटिंग पॉइंट नंबरवर केलेल्या बेरीज किंवा वजाबाकी ऑपरेशनचे परिणाम रजिस्टर AC मध्ये संग्रहित करतात. बेरीज किंवा वजाबाकी चार स्टेप्स मध्ये केली जाऊ शकते.

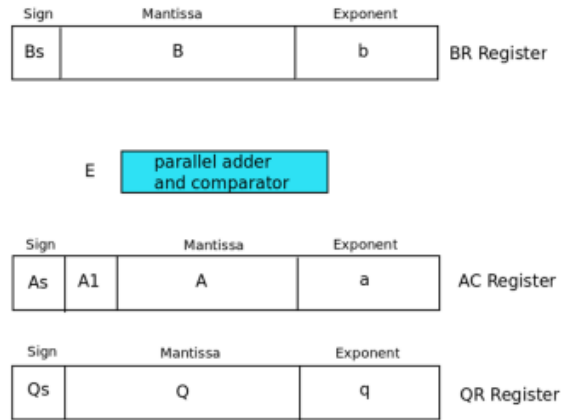
S1: शून्ये तपासा.

S2: मान्टिसास संरेखित करा.

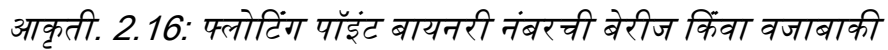
S3: मान्टिसावर बेरीज किंवा वजाबाकी करा.

S4: परिणाम सामान्यीकरण.

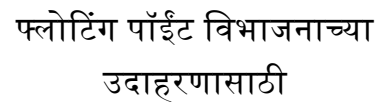
रजिस्टर AC, BR. आणि QR मध्ये एक फ्लोटिंग-पॉइंट क्रमांक आहेत, ज्यामध्ये साईन, मान्टिसा आणि घातांक असे तीन घटक असतात. AC रजिस्टर मधील फ्लोटिंग पॉइंट नंबर साईन चे प्रतिनिधित्व करत असल्याने, A1 मान्टिसामधील सर्वात महत्त्वपूर्ण बिट दर्शवितो आणि फ्लोटिंग पॉइंट नंबरचे घातांक दर्शवितो. आकृती 2.16 मध्ये दर्शविल्याप्रमाणे रजिस्टर AC आणि BR हे बेरीज किंवा वजाबाकीसाठी वापरले जातात. आकृती 2.17 मध्ये पाहिल्याप्रमाणे फ्लोटिंग पॉइंट नंबरच्या गुणाकारात रजिस्टर QR चा वापर केला गेला आहे.

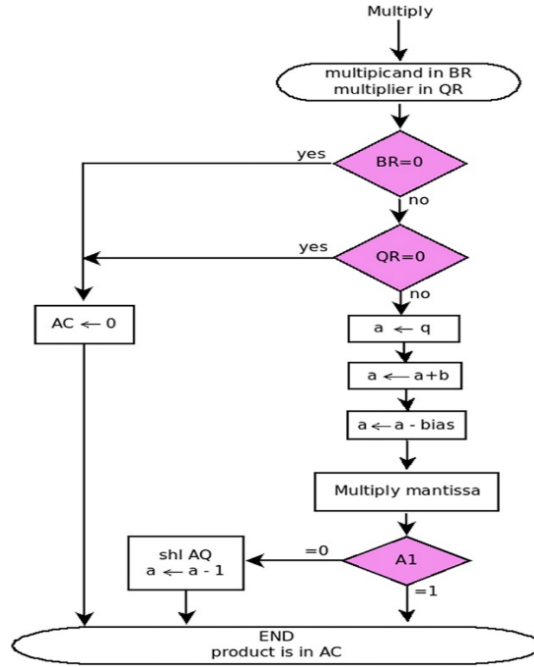


आकृती. 2.15: फ्लोटिंग पॉइंट अरिथमेटिक ऑपरेशन्ससाठी रजिस्टर



S4: परिणाम सामान्यीकरण.





आकृती. 2.17 फ्लोटिंग पॉइंट बायनरी नंबर्सचा गुणाकार

आकृती. 2.17 मध्ये या स्टेप्स चे तपशीलवार वर्णन करते. फ्लोटिंग पॉइंट नंबर्सचा गुणाकार करण्यापूर्वी, ते शून्य आहे की नाही हे निर्धारित करणे आवश्यक आहे. कोणतीही संख्या शून्य असल्यास परिणाम 0 आहे.

जर दोन्ही संख्या नॉन झीरो असतील, तर मान्टिसासच्या गुणाकार आणि घातांकांच्या जोडणीचा गुणाकार AC रजिस्टरमध्ये ठेवला जातो. जर अंतिम परिणाम सामान्य स्वरूपात (NORMLIZED) नसेल, तर घातांक वाढवावा किंवा कमी करावा यावर अवलंबून, परिणाम डावीकडे किंवा उजवीकडे हलवून तो सामान्य केला जातो.

3. भागाकार (Division)

फ्लोटिंग पॉइंट नंबर्सची भागाकार खालील स्टेप वापरून केली जाऊ शकते:

S1: शून्ये तपासा.

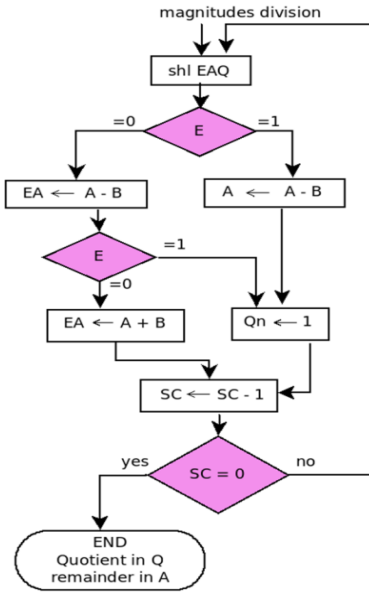
S2: रजिस्टर इनीशीलाईझ करणे आणि साइन व्हॅल्यूमापन करणे.

S3: डिविडेंड एकरूप करणे.

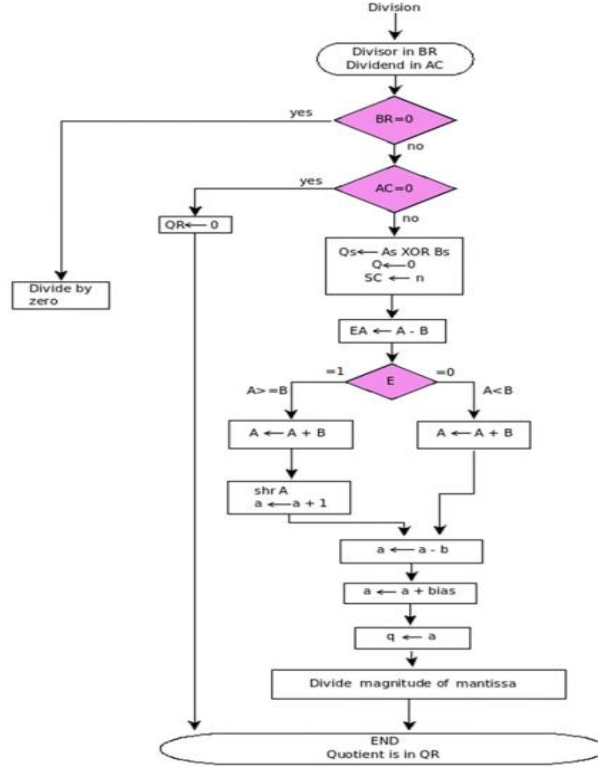
S4: घातांक वजा करणे.

S5: मान्टिसा चा भागाकार करणे.

आकृती. 2.19 फ्लोटिंग पॉइंट नंबर भागाकार करण्यासाठी आवश्यक कार्यपद्धती तपशीलवार वर्णन करते. डिव्हिजन ऑपरेशन्समध्ये, डिव्हायझर शून्य आहे की नाही हे प्रथम फिक्स्ड केले पाहिजे. डिव्हायझर शून्य असल्यास डिव्हाइड ओव्हरफ्लो होईल. जर डिव्हासर शून्य नसेल, तर आकृती 2.18 मध्ये दर्शविल्याप्रमाणे, मान्टिसास भागाकार करण्यापूर्वी घातांक वजा केल्या जातात.



आकृती. 2.18 माग्नीट्यूड भागाकार



आकृती. 2.19 फ्लोटिंग पॉइंट बायनरी नंबर्स चा भागाकार

2.6 अरीथमेटिक पाइपलाइन

पाइपलाइन अरीथमेटिक युनिट बहुतेकदा सुपरकंप्यूटरमध्ये दिसतात. त्यांचा वापर फ्लोटिंग-पॉइंट ऑपरेशन्स, फिक्स्ड-पॉइंट मल्टिप्लिकेशन आणि वैज्ञानिक परिस्थितीत आढळणाऱ्या इतर गणना पूर्ण करण्यासाठी केला जातो. पाइपलाइन मल्टिप्लायर हा केवळ विशिष्ट ऍडर्ससह एक अर्रे मल्टिप्लायर आहे जो आंशिक उत्पादनांमध्ये कॅरी प्रसार वेळ कमी करतो.

आकृती 2.20 मध्ये दर्शविल्याप्रमाणे, फ्लोटिंग-पॉइंट ऑपरेशन्स सहजपणे उप-ऑपरेशन्समध्ये विभागल्या जाऊ शकतात. फ्लोटिंग-पॉइंट अँडर पाइपलाइन ला पाइपलाइन युनिटसाठी इनपुट म्हणून दोन सामान्य फ्लोटिंग-पॉइंट बायनरी व्हॅल्यू मिळतात.

$$W = A \times 2^a$$

$$Z = B \times 2^b$$

मान्तिसा (A आणि B अक्षरांनी दर्शविलेले) आणि एक्सपोनॅन्ट (A आणि B अक्षरांनी दर्शविलेले) अनुक्रमे A आणि B ही अभिव्यक्ती तयार करतात. आकृती 2.20 मध्ये दर्शविल्याप्रमाणे, फ्लोटिंग-पॉइंट नंबरसह बेरीज आणि वजाबाकी करण्याचे ऑपरेशन चार विभागांमध्ये विभागले जाऊ शकते.

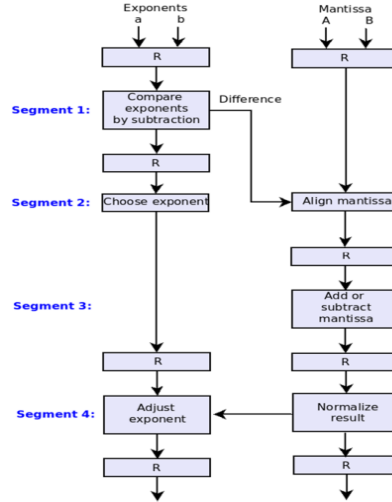
इंटरमीडिएट माहितीच्या तात्पुरत्या साठवणीसाठी, रजिस्टर R अक्षराने दर्शविलेल्या विभागाशी जोडले जाऊ शकतात. चार विभागांमध्ये खालील सब ऑपरेशन चा समावेश आहे:

S1: एक्सपोनॅन्ट ची तुलना करणे.

S2: मान्टिसास एकरूप करणे.

S3: मान्टिसावर बेरीज /वजाबाकी ऑपरेशन करणे.

S4: परिणाम नॉर्मलाइझेशन करणे.



आकृती. 2.20: फ्लोटिंग पॉइंट
अंडरची पाईपलाईन अंमलबजावणी

प्रथम, आकृती 2.20 मध्ये दर्शविल्याप्रमाणे एक्सपोनंट वजा करा. एक्सपोनंट चा फरक हा खालच्या एक्सपोनंट चा मांटिसा किती वेळा उजवीकडे किंवा डावीकडे हलवला पाहिजे हे ठरवतो. हे मांटिसास अलाइन करते. तिसरी स्टेप्स म्हणजे मान्टिसा जोडले जातात किंवा वजा केले जातात. शेवटचा निकाल सामान्य करा.

ओव्हरफ्लो एक्सपोनंट एकाने वाढवतात आणि बेरीज किंवा वजाबाकी मान्टिसाला उजवीकडे हलवतात. मान्टिसामधील लिडिंग शून्यांचे प्रमाण एक्सपोनंट कमी होणे आणि अंडरफ्लो असताना मान्टिसा किती दूर डावीकडे सरकते हे निर्धारित करते.



स्कॅन करा

उदाहरणासह अरीथमेटिक
पाईपलाईन समजून घेण्यासाठी

2.7 इंस्ट्रक्शन पाइपलाईन

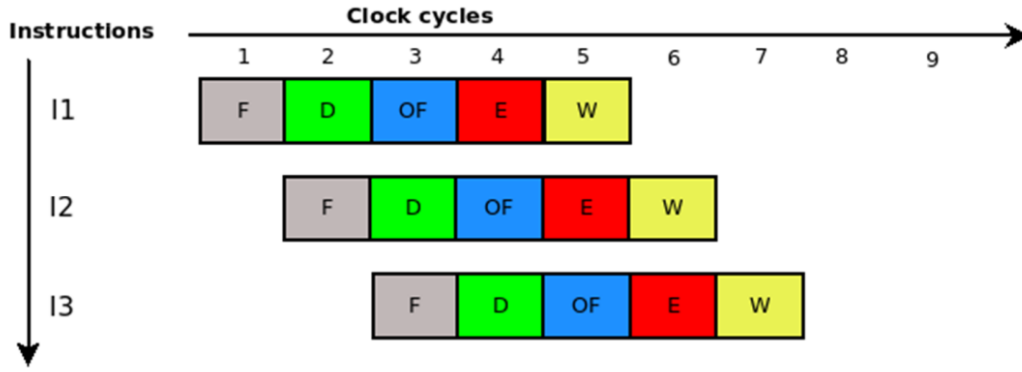
इंस्ट्रक्शन पाईपलाईनिंग मध्ये एकाच प्रोसेसर क्लॉक सायकल मध्ये अनेक इंस्ट्रक्शन अंमलात आणल्या जातात. दुसरीकडे, नॉन-पाईपलाईन आर्किटेक्चर मध्ये एकामागून एक अनुक्रमे इंस्ट्रक्शन अंमलात आणल्या जातात. आकृती. 2.21 पाच टप्प्यांसह पाईपलाईन प्रोसेसरवर तीन इंस्ट्रक्शन दर्शविते (fetch (F) decode (D) operand fetch (OF) execute (E) आणि write back (W)) प्रत्येक स्टेट एकाच प्रोसेसर क्लॉक सायकल मध्ये पूर्ण होते. पाइपलाईनमधील



स्कॅन करा

इंस्ट्रक्शन पाइपलाईन आणि
परफॉर्मन्स बद्दल अधिक
जाणून घेण्यासाठी

तीन इंस्ट्रक्शन कार्यान्वित करण्यासाठी 7 प्रोसेसर क्लॉक सायकल ची आवश्यकता असते. पाच टप्प्यांसह तीन इंस्ट्रक्शन वर प्रोसेस करण्यासाठी पाईपलाईनच्या अनुपस्थितीत 15 प्रोसेसर क्लॉक सायकल लागतात.



आकृती. 2.21: इंस्ट्रक्शन पाईपलाईन अंमलबजावणी

समजा K स्टेज सह N इंस्ट्रक्शन आहेत आणि प्रत्येक इंस्ट्रक्शन एका स्टेज दरम्यान T टाइम युनिट जास्तीत जास्त वेळ डीले आहे. पाईपलाईनसह आणि त्याशिवाय इंस्ट्रक्शन अंमलात आणण्यासाठी लागणारा वेळ अशा प्रकारे मोजला जाऊ शकतो.

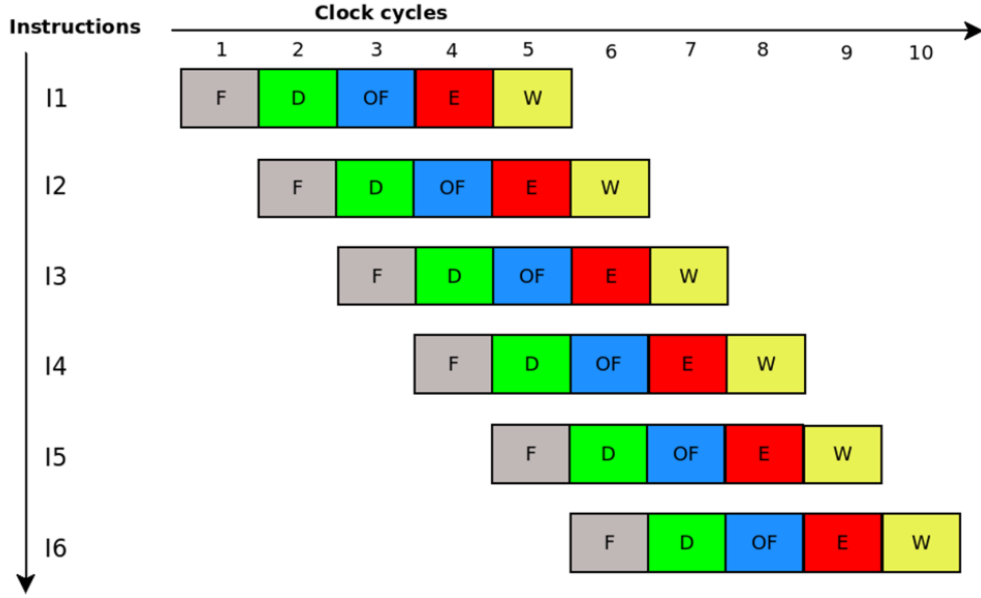
- पाइपलाइनमध्ये N इंस्ट्रक्शन कार्यान्वित करण्यासाठी लागणारा वेळ = $(N + K - 1) \times T$, आणि K स्टेज ची संख्या, N इंस्ट्रक्शन ची एकूण संख्या आणि T दोन स्टेज मधला टाइम डीले आहे.
- नॉन-पाइपलाइन प्रोसेसरमध्ये N इंस्ट्रक्शन कार्यान्वित करण्यासाठी लागणारा वेळ = $N \times K \times T$.
- स्पीड अप फॅक्टर = $N \times K / (K + N - 1)$

इंस्ट्रक्शन च्या अंमलबजावणीदरम्यान उद्भवू शकणारे तीन प्रकारचे पाईपलाईन हझार्ड आहेत.

2.7.1 रेसोर्स किंवा संरचनात्मक हझार्ड:

पाइपलाइनमध्ये अनेक इंस्ट्रक्शन असल्यास संरचनात्मक हझार्ड काही इंस्ट्रक्शन पॅरलल एक्झिक्युशन होण्या पासून रोखू शकतात. उदाहरणार्थ, मुख्य मेमोरी मध्ये फक्त एक पोर्ट असतो. प्रोसेसरने प्रत्येक रीड किंवा राइट कार्यासाठी समान पोर्ट वापरणे आवश्यक आहे. कल्पना करा की एकाच प्रोसेसर क्लॉक सायकल मध्ये, अनेक इंस्ट्रक्शन एकाच प्रायमरी मेमोरी रेसोर्स ला ऍक्सेस करतात. हार्डवेअर रेसोर्स च्या मर्यादांमुळे, म्हणजे एकच प्रायमरी मेमोरी, केवळ एकच इंस्ट्रक्शन एकाच क्लॉक सायकल मध्ये एक्झिक्युशन पूर्ण करू शकतात. अशा रेसोर्स च्या मर्यादा संरचनात्मक धोके (STRUCTURAL HAZARD) म्हणतात.

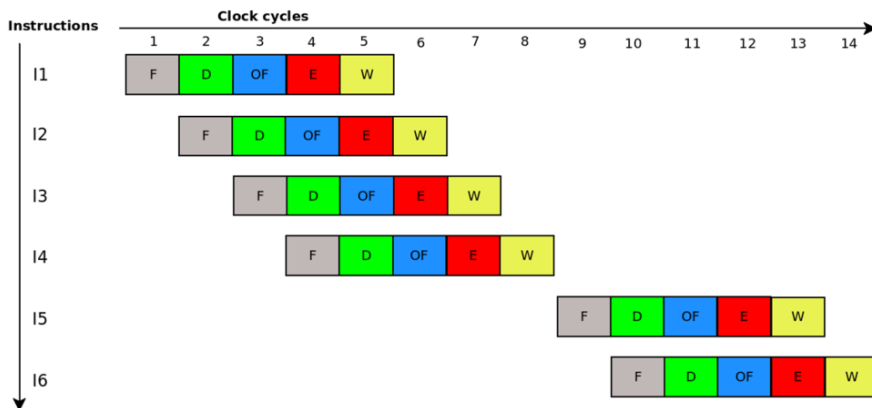
आकृती 2.22 मध्ये., इंस्ट्रक्शन I1 सायकल 5 मध्ये मेमोरी राइटबॅक करते, तर इंस्ट्रक्शन I5 सायकल 5 मध्ये फेच ऑपरेशन करतात. त्याचप्रमाणे इंस्ट्रक्शन I2 राइट ऑपरेशन करण्यासाठी मेमोरी ऍक्सेस करतात, त्याच वेळी इंस्ट्रक्शन I6 सहाय्या क्लॉक सायकल दरम्यान इंस्ट्रक्शन फेच करण्याचे कार्य करते. एकाच पोर्ट मेमोरीवर एकाच वेळी रीड आणि राइट ऑपरेशन करणे व्यवहार्य नाही, अशा प्रकारे पाचव्या आणि सहाव्या क्लॉक सायकल दरम्यान, डेटा राइट आणि इंस्ट्रक्शन रीड करण्या करीता मेमोरी ला ऍक्सेस केला जातो.



आकृती. 2.22: रेसोर्स किंवा संरचनात्मक हझार्ड

एकाच रेसोर्स (memory) वर वेगवेगळ्या ऑपरेशन्स करणार्या दोनपेक्षा जास्त इन्स्ट्रक्शन असल्यास पाइपलाइनमध्ये समस्या निर्माण होतात. अशा प्रकारच्या हझार्ड ला संरचनात्मक धोके (STRUCTURAL HAZARD) असे म्हणतात.

उपलब्ध रेसोर्स चे प्रमाण वाढवून या प्रकारचा हझार्ड कमी केला जाऊ शकतात. संरचनात्मक धोके टाळण्यासाठी, आधुनिक प्रोसेसर इन्स्ट्रक्शन आणि डेटा स्टोर करण्यासाठी स्वतंत्र मेमोरी वापरतात. वैकल्पिकरित्या, इन्स्ट्रक्शन सेट अंमलात आणून, काही क्लॉक साठी पाईपलाईन थांबवून, इन्स्ट्रक्शन सेट पुन्हा अंमलात आणून आणि प्रोग्राम पूर्ण होईपर्यंत प्रक्रिया सुरू ठेवून समस्या टाळली जाऊ शकते. हा उपाय आकृती 2.23 मध्ये दर्शविला गेला आहे. पहिल्या तीन स्टेप्स पूर्ण केल्यानंतर, इतर स्टेप्स क्रमाने अंमलात आणणे आवश्यक आहे.



आकृती. 2.23 संरचनात्मक/रेसोर्स हझार्ड सोलुशन

2.7.2 डेटा हझार्ड (Data Hazards):

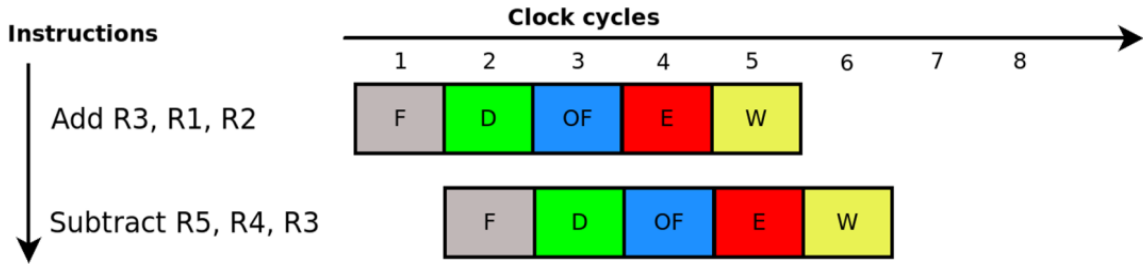
जेव्हा पाइपलाइनमधील दोन इन्स्ट्रक्शन एकाच वेळी कार्यान्वित होतात, तेव्हा डेटा हझार्ड उद्भवतात. जर दुसऱ्या इन्स्ट्रक्शन च्या अंमलबजावणीचा परिणाम पहिल्या इन्स्ट्रक्शन च्या परिणामावर अवलंबून असेल, तर डेटा हझार्ड उद्भवतो. डेटा हझार्ड चे वर्गीकरण RAW हझार्ड, WAR हझार्ड आणि WAW हझार्ड असे केले जाऊ शकते.

1. RAW हझार्ड:

रीड आफ्टर राईट' (RAW) हे खरी डिपेंडन्शी आहे. सिक्वेंशियल एक्सिक्युशन मध्ये कोणतीही अडचण नाही. जर इन्स्ट्रक्शन एकाच वेळी अंमलात आणल्या गेल्या, तर त्या कोणत्याही सिक्वेन्स ने अंमलात आणल्या जाऊ शकतात, परिणामी चुकीचा रिझल्ट किंवा आउटपुट होऊ शकतो. उदाहरणार्थ, दिलेल्या I1 आणि I2 या दोन्ही इन्स्ट्रक्शन शेअर्ड रजिस्टर R3 वापरतात. त्यामुळे RAW चा धोका येथे तपासला पाहिजे.

I1: Add R3, R1, R2

I2: Subtract R5, R4, R3

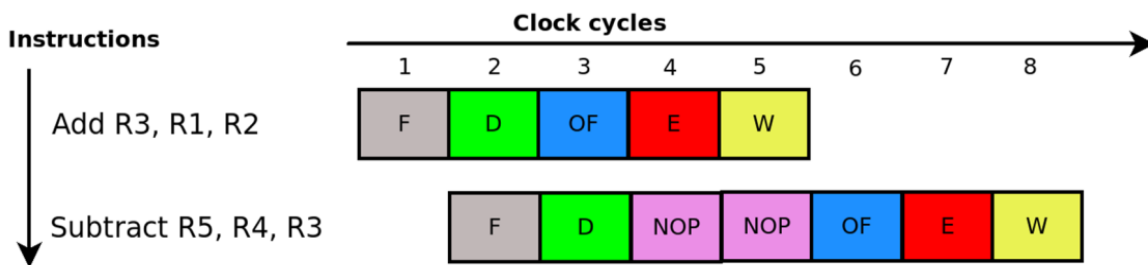


आकृती. 2.24: इन्स्ट्रक्शन पाईपलाईन अंमलबजावणीमध्ये रीड आफ्टर राईट (RAW) हझार्ड घटना

ज्या प्रकारे आकृती. 2.24 दर्शविले आहे. जेव्हा इन्स्ट्रक्शन I1 डीकोड टप्प्यात आहे, इन्स्ट्रक्शन I2 Fetch टप्प्यात आहे. मेमोरी राईट स्टेप नंतर ऑपरेंड R3 चे व्हॅल्यू बदलले जाईल. सध्या, जुन्या R3 व्हॅल्यूचा वापर करून इन्स्ट्रक्शन सबट्रॅक्ट कार्यान्वित केले जाते. कारण I2 इन्स्ट्रक्शन ची अंमलबजावणी पूर्णपणे I1 इन्स्ट्रक्शन पूर्ण करण्यावर अवलंबून असते. जेव्हा I2 इन्स्ट्रक्शन I1 इन्स्ट्रक्शन रीड कार्य पूर्ण होण्यापूर्वी कार्यान्वित केली जाते, तेव्हा रीड आफ्टर राईट (RAW) हझार्ड उद्भवतो.



स्कॅन करा
इन्स्ट्रक्शन पुनर्रचना जाणून घेण्यासाठी



आकृती. 2.25: RAW इन्स्ट्रक्शन हझार्ड चे उपाय

RAW चा हझार्ड दूर करण्यासाठी, आकृती 2.25 मध्ये दर्शविल्याप्रमाणे I1 रीड ऑपरेशन पूर्ण झाल्यानंतर, 6 व्या क्लॉक सायकल मध्ये इन्स्ट्रक्शन I2 चे ऑपरेंड आणल्या जातील.

एनओपी NOP (नो ऑपरेशन) समाविष्ट केल्यामुळे, पाईपलाईन स्लो होतात. RAW चे हझार्ड आणखी दोन पर्याय वापरून सोडवले जाऊ शकतात. प्रथम, इन्स्ट्रक्शन ची अशी पुनर्रचना करा की अवलंबून नसलेल्या इन्स्ट्रक्शन नॉन डिपेंडेंट इन्स्ट्रक्शन च्या दरम्यान ठेवल्या जातील. कंपायलर इन्स्ट्रक्शन ची पुनर्रचना करू शकतो. दुसरे म्हणजे मोस्ट रीसेन्ट ऑपरेंड ऑपरेंड चे व्हॅल्यू पाइपलाइन टप्प्यांवरील डिपेंडेंट इन्स्ट्रक्शन कडे पाठवले जाऊ शकते.



स्कॅन करा
ऑपरेंड फॉरवर्डिंग
जाणून घेण्यासाठी

उदाहरण 2.6

तुम्ही पाइपलाइनमध्ये Add R3, R7, R1 आणि Subtract R5, R8, R3 इन्स्ट्रक्शन कसे एक्सिक्युट कराल? ऑपरेंड व्हॅल्यू खालीलप्रमाणे आहेत: R1 = 20, R7 = 30, R3 = 10, आणि R8 = 400.

उपाय: इन्स्ट्रक्शन I1 साठी, R1 हे 20 वर सेट केले आहे, R7 हे 30 वर सेट केले आहे आणि इन्स्ट्रक्शन I1 (Add) चा रिझल्ट 50 आहे, जो रजिस्टर R3 मध्ये ठेवला आहे आणि क्लॉक सायकल 5 दरम्यान अपडेट केला आहे. जर I2 इन्स्ट्रक्शनला क्लॉक सायकल मध्ये R3 ऑपरेंडची आवश्यकता असेल आणि R3 ऑपरेंड चौथ्या क्लॉक सायकल मध्ये आणले असेल, तर इन्स्ट्रक्शन I2, R3 = 10 असे रीड. हे ऑपरेंडचे जुने व्हॅल्यू आहे, परिणामी विसंगत डेटा तयार होतो. अंतिम परिणाम $R5 = R8 - R3 = 400 - 10 = 390$ आहे. जेव्हा दोन्ही इन्स्ट्रक्शन अनुक्रमे (एकापाठोपाठ एक) एक्सिक्युट केल्या जातात तेव्हा परिणाम $R3 = R1 + R7 = 20 + 30 = 50$; $R5 = R8 - R3 = 400 - 50 = 350$. ऑपरेंड R5 साठी 350 हे व्हॅल्यू बरोबर आहे.

रिझल्ट मधील ही विसंगती RAW च्या हझार्ड मुळे होतात. पॅरलल एक्सिक्युशन दरम्यान RAW चे हझार्ड टाळण्यासाठी, इन्स्ट्रक्शन I2 चे ऑपरेंड सहाव्या क्लॉक सायकल मध्ये आणले पाहिजेत, तर प्रोसेसर चौथ्या आणि पाचव्या क्लॉक सायकल मध्ये निष्क्रिय/थांबलेला/NOP (ऑपरेशन नाही) राहतात. परिणामी, I2 इन्स्ट्रक्शन अपडेटेड ऑपरेंड प्राप्त करते ($R3 = 50$) R8 400 आहे आणि प्रोग्राम योग्य रिझल्ट ($R5 = 350$) तयार करतात.

2. WAR हझार्ड:

WAR हझार्ड ला अँटी डिपेंडन्सी असेही म्हणतात. हे हझार्ड तेव्हा उद्भवतात जेव्हा एकाच वेळी इन्स्ट्रक्शन एक्सिक्युट होतात आणि आधीच्या इन्स्ट्रक्शन द्वारे रीड केल्या नंतर ताबडतोब एका इन्स्ट्रक्शन चे डेस्टिनेशन /आउटपुट रजिस्टर वापरले जातात.

I1: Mul R6, R7, R8 /R6 ← R7 x R8/

I2: Add R8, R9, R10 /R8 ← R9 + R10/

उदाहरणार्थ, जर हे इन्स्ट्रक्शन I1 आणि I2 अनुक्रमे अंमलात आणल्या गेले तर कोणतीही अडचण नाही. तथापि, जेव्हा I1 आणि I2 एकाच वेळी कार्यान्वित केले जातात, तेव्हा WAR हजार्ड उद्भवू शकतो.

उदाहरण 2.7

तुम्ही पाईपलाईनमध्ये Mul R6, R7, R8 आणि Add R8, R9, R10 इन्स्ट्रक्शन कसे अंमलात आणाल? ऑपरेंड व्हॅल्यू खालीलप्रमाणे आहेत: R7 = 20, R8 = 10, R9 = 30 आणि R10 = 50.

चार किंवा पाच स्टेजच्या पाईपलाईनमध्ये 'WAR' हा हजार्ड कमी असण्याची शक्यता असते.

$$I1: R6(200) \leftarrow R7(20) \times R8(10)$$

$$I2: R8(80) \leftarrow R9(30) + R10(50)$$

R8 हे रजिस्टर दोन्ही इन्स्ट्रक्शन द्वारे वापरले जाते. जर I2 इन्स्ट्रक्शन, I1 इन्स्ट्रक्शन त्याच्या ऑपरेंडला आणण्यापूर्वी अंमलबजावणी पूर्ण करते, तर WAR धोका उद्भवू शकतो. तथापि, जर I1 नंतर I2 कार्यान्वित केला गेला तर कोणतीही समस्या उद्भवणार नाही. परंतु I2 इन्स्ट्रक्शन राइट केल्या नंतर, I1 इन्स्ट्रक्शन अंमलात आणली जातात. युद्धाचा धोका आहे कारण I1 इन्स्ट्रक्शन रजिस्टर R8 चे सुधारित व्हॅल्यू रीड केले जातात.

$$I1: R6(1600) \leftarrow R7(20) \times R8(80)$$

$$I2: R8(80) \leftarrow R9(30) + R10(50)$$

या कारणास्तव, प्रोग्राम R6 साठी चुकीचे व्हॅल्यू देईल (1600). तथापि, R8 या रजिस्टर चे नाव बदलल्याने ही समस्या सुटू शकते.

3. WAW हजार्ड:

इन्स्ट्रक्शन एकाच वेळी अंमलात आणल्या गेल्यास 'राइट आफ्टर राइट' (WAW) हा हजार्ड उद्भवतो. जर I1 आणि I2 हे दोन इन्स्ट्रक्शन एकाच वेळी अंमलात आणल्या गेले, तर इन्स्ट्रक्शन I1 ला डीले होईल आणि इन्स्ट्रक्शन I2 पूर्ण होईल असा अंदाज आहे. या परिस्थितीत, WAW चा हजार्ड उद्भवतो. I1 आणि I2 या दोन इन्स्ट्रक्शन एकाच आउटपुट रजिस्टर R8 वर लिहिण्याचा परिणाम म्हणून ही परिस्थिती उद्भवते.

$$I1: \text{Add } R8, R6, R7 \text{ / } R8 \leftarrow R6 + R7$$

$$I2: \text{Add } R8, R9, R10 \text{ / } R8 \leftarrow R9 + R10$$

उदाहरण 2.8

तुम्ही पाईपलाईनमध्ये R8, R6, R12 आणि R8, R9, R10 इन्स्ट्रक्शन कशा एक्सिक्युट कराल? ऑपरेंड व्हॅल्यू खालीलप्रमाणे आहेत: R6 = 50, R12 = 50, R9 = 100 आणि R10 = 150.



स्कॅन करा

रजिस्टर रेनेमिंग मध्ये
रजिस्टर ऑलोकेशन टेबल
बद्दल जाणून घेण्यासाठी

उपाय: येथे, $R6 = 50$, $R12 = 50$, $R9 = 100$, आणि $R10 = 150$. जर I1 आणि I2 इन्स्ट्रक्शन पाईपलाईनमध्ये अंमलात आणल्या गेल्या, तर I1 आणि I2 पूर्ण झाल्यानंतर R8 दुसऱ्या इन्स्ट्रक्शन अंमलात आणण्याचा निकाल राखून ठेवेल. या प्रकरणात, समस्या नाही आहे.

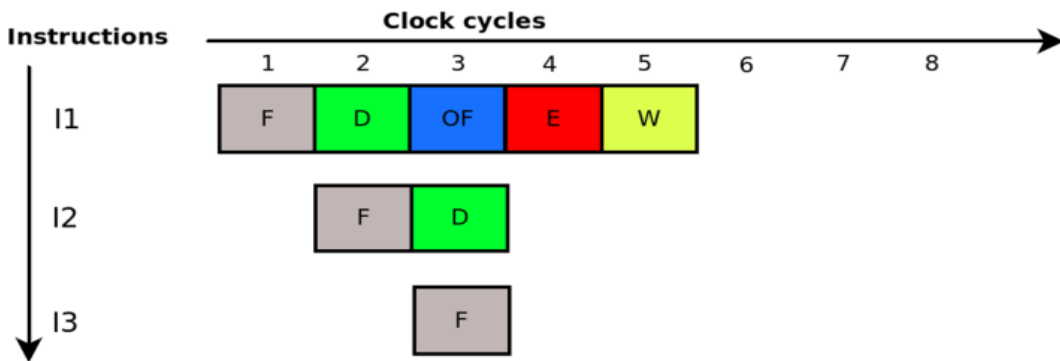
$$I1: R8(100) \leftarrow R6(50) + R12(50)$$

$$I2: R8(250) \leftarrow R9(100) + R10(150)$$

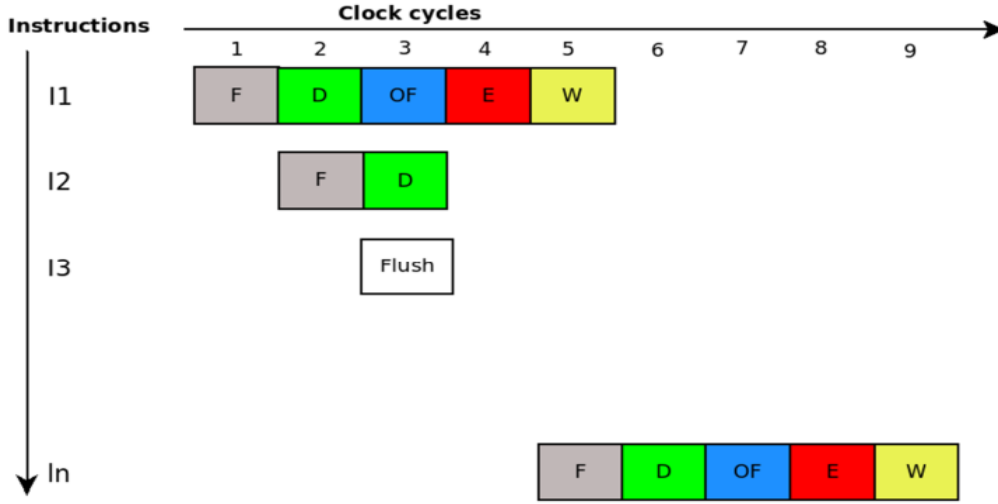
परंतु, I1 इन्स्ट्रक्शन अंमलबजावणीस विलंबित करते, I2 इन्स्ट्रक्शन त्याची अंमलबजावणी पूर्ण करते आणि R8 (250) रजिस्टर I2 निकाल स्टोर करते. I1 इन्स्ट्रक्शन नी अखेरीस त्याची अंमलबजावणी पूर्ण केली आणि परिणाम R8 (100) रजिस्टर वर राइट केला गेला. याला WAW हज्जार्ड असे म्हणतात. डेस्टिनेशन रजिस्टर चे नाव बदलून हा धोका दूर केला जाऊ शकतो.

2.7.3 कंट्रोल हज्जार्ड

कंट्रोल हज्जार्ड ब्रांच इन्स्ट्रक्शन शी संबंधित आहेत. चला कल्पना करूया की पाईपलाईनमध्ये तीन इन्स्ट्रक्शन (I1, I2 आणि I3) आहेत. इन्स्ट्रक्शन I2 ही एक ब्रांच इन्स्ट्रक्शन आहे असे गृहीत धरून, डिकोडिंग टप्प्यानंतरपर्यंत प्रोसेसरला या वस्तुस्थितीची माहिती नसते. ब्रांच एंड्रेस प्रोग्राम काउंटरमध्ये (PC) भरला जातो आणि प्रोसेसर नंतर ब्रांच च्या एंड्रेस द्वारे दिलेल्या इन्स्ट्रक्शन अंमलात आणतात. पाईपलाईन इन्स्ट्रक्शन I1, I2 आणि I3 आकृती 2.26 मध्ये दर्शविल्या आहेत. तिसऱ्या क्लॉक सायकल दरम्यान इन्स्ट्रक्शन I2 डीकोड केली जाते आणि प्रोसेसर ती ब्रांच इन्स्ट्रक्शन म्हणून ओळखतो. तथापि, इन्स्ट्रक्शन I2 च्या डीकोडिंग दरम्यान इन्स्ट्रक्शन I3 आधीच मेमोरीमधून पुनर्प्राप्त केली गेली आहे. म्हणून, पाईपलाईन इन्स्ट्रक्शन चा हा चुकीचा क्रम आहे. याला कंट्रोल हज्जार्ड असे म्हणतात.



आकृती. 2.26: कंट्रोल हज्जार्ड परिस्थिती



आकृती. 2.27: कंट्रोल हार्डार्ड सोलुशन

कंट्रोल हार्डार्ड च्या दरम्यान पाईपलाईनमध्ये नवीन इन्स्ट्रक्शन I3 घातल्यास, प्रोसेसर पाईपलाईन फ्लश करेल. आकृती 2.27 मध्ये दर्शविल्याप्रमाणे ब्रांच इन्स्ट्रक्शन ऍड्रेस संगणकात भरला जातो. हे बिनशर्त ब्रांच इन्स्ट्रक्शन कंडिशनल तपासणी न करता विशिष्ट ठिकाणी जम्प करतात. उदाहरणार्थ, जम्प 2000 इन्स्ट्रक्शन 2000 ऍड्रेस च्या ठिकाणी डारेक्टिव अंमलबजावणी सुरू करते. याउलट, कंडिशनल ब्रांच इन्स्ट्रक्शन स्थिती तपासतात आणि कंडिशन खरी असल्यास ब्रांच इन्स्ट्रक्शन अंमलात आणतात (if, if-else, loops). त्याव्यतिरिक्त, इन्स्ट्रक्शन प्राप्त झालेल्या क्रमानुसार अंमलात आणल्या जातात.

फ्लशिंग व्यतिरिक्त, ब्रांच प्रेडिक्शन प्रोसेसरच्या पाईपलाईन अंमलबजावणीचे फायदे जास्तीत जास्त करण्यासाठी वापरले जातात. पाइपलाईनच्या रचनेत, हार्डवेअर सर्किट ब्रांच घेतली जाईल की नाही याचा अंदाज लावण्याचा प्रयत्न करते. जर एखादी ब्रांच कार्यान्वित होत असताना पाईपलाईनमध्ये ओळखली गेली असेल, तर प्रोसेसर काही क्लॉक सायकल साठी (स्टॉल किंवा बबल) प्रतीक्षा करतो. यामुळे पाईपलाईनची कार्यक्षमता कमी होते. अंमलबजावणीपूर्वी ब्रांच इन्स्ट्रक्शन साठी अंदाज लावल्याने पाईपलाईनिंगची गती वाढते. ब्रांच प्रेडिक्शन लावण्यासाठी दोन पद्धती आहेत: स्टॅटिक आणि डायनॅमिक ब्रांच प्रेडिक्शन.

1. स्टॅटिक ब्रांच प्रेडिक्शन

स्टॅटिक ब्रांच प्रेडिक्शन असे गृहीत धरतो की कोणतीही ब्रांच घेतली जाणार नाही. ते पुढील इन्स्ट्रक्शन चे ऍड्रेस क्रमाने आणते. प्रत्येक चुकीच्या प्रेडिक्शन साठी ब्रांच घेणे आवश्यक आहे. ब्रांच टारगेट इन्स्ट्रक्शन मागील इन्स्ट्रक्शन पुनर्स्थित करते. चुकीच्या अंदाजामुळे पाईपलाईन क्लॉक सायकल ची पेनल्टी लागतात.

अंदाज अचूक असल्यास ब्रांच इन्स्ट्रक्शन च्या अंमलबजावणीला गती येतात. जेव्हा ब्रांच कंडिशनल पूर्ण होते, तेव्हा नेहमी तोच पर्याय (घेतला नाही असे गृहीत धरून) निवडला जातो.

2. डायनॅमिक ब्रांच प्रेडिक्शन

डायनॅमिक ब्रांच प्रेडिक्शन हे इतिहासावर आधारित प्रेडिक्शन तंत्र आहे. ती आधीच्या ब्रांच



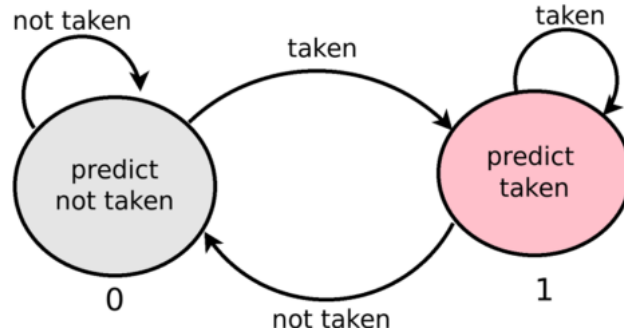
स्कॅन करा

डायनॅमिक ब्रांच प्रेडिक्शन
समजून घेण्यासाठी

च्या माहितीसह ब्रांच हिस्टरी टेबल (बीएचटी) ठेवते. एक किंवा दोन बिट्स वापरून डायनॅमिक ब्रांच प्रेडिक्शन केले जाऊ शकतो.

3. 1-बिट डायनॅमिक ब्रांच प्रेडिक्शन

ब्रांच हिस्टरी टेबल मध्ये ब्रांच बदल 1 बिट माहिती आहे. जी 1 बिट डायनॅमिक ब्रांच प्रेडिक्शन (1) घेतली जाईल किंवा (0) घेतली जाणार नाही असा अंदाज आहे. त्यानंतर प्रोसेसर या आधारावर टारगेट ब्रांच तून किंवा अनुक्रमांमधून त्यानंतर चे इन्स्ट्रक्शन पुनर्प्राप्त करतो.



आकृती. 2.28: 1-बिट डायनॅमिक ब्रांच प्रेडिक्शनची स्थिती आकृती

अनुक्रमे घेतलेले प्रेडिक्ट बायनरी व्हॅल्यू 1 सूचित करते आणि न घेतलेले प्रेडिक्ट बायनरी व्हॅल्यू 0 आहे. आकृती 2.28 मध्ये ब्रांच प्रेडिक्शन टेबल पूर्व ब्रांच च्या माहितीचा मागोवा आहेत. पूर्वीच्या माहितीच्या आधारे हे फिक्स्ड केले गेले आहे की कोणत्याही एका ब्रांच ने घेतले आहे किंवा घेतले नाही.

उदाहरण 2.9

दिलेल्या प्रोग्रामसाठी 1-बिट ब्रांच प्रेडिक्टर किती वेळा ब्रांच चा योग्य प्रेडिक्शन लावतो? ब्रांच ने प्रारंभिक स्थितीत घेतले असावे असे मानले जाते.

```

int a=0;
while(a<5)
{
    if(a%2==0){
        branch instructions
    }
    a++;
}
  
```

उपाय:

लूप खालील प्रमाणे रन होणार. आकृती 2.29 हार्डवेअर प्रेडिक्शन आणि वास्तविक प्रेडिक्शन दर्शवितो. कोणती फांदी घ्यायची ही सुरुवातीची अट आहे. तर ब्रांच $a = 0$ वर घेतली आहे.

- हार्डवेअर असे प्रेडिक्ट करते की ब्रांच $a = 1$ वर घेतली जाते, परंतु ती ब्रांच घेतली जाऊ नये.
- हार्डवेअर असे प्रेडिक्ट करते की ब्रांच $a = 2$ वर घेतली जात नाही, परंतु ती ब्रांच घेणे आवश्यक आहेत.
- $a = 3$ वर, हार्डवेअर असे गृहीत धरते की ब्रांच घेतली गेली आहे, परंतु ती घेतली गेली नाही.
- हार्डवेअर असे गृहीत धरते की ब्रांच $a = 4$ वर घेतली जात नाही, परंतु ती घेतली जातात.

	a=0	a=1	a=2	a=3	a=4
Actual prediction	T	N	T	N	T
Hardware prediction	T	T	N	T	N
	correct	wrong	wrong	wrong	wrong

आकृती 2.29 हार्डवेअर प्रेडिक्शन आणि वास्तविक प्रेडिक्शन

ज्या प्रकारे आकृती 2.29 दर्शविले आहे. $a = 0$, $a = 2$, $a = 4$ ब्रँचिंग घेतले आहे, तर शाखेला $a = 1$, $a = 3$ असे ब्रँचिंग नाही घेतले आहे. तथापि, $a = 0$ वरील ब्रांच पूर्व ब्रांच च्या माहितीवर आधारित हार्डवेअर अंदाजामुळे घेतली गेली आहे. वास्तविक अंदाजात, $a = 1$ वरील ब्रांच घेत नाही, परंतु हार्डवेअर अंदाजात, $a = 1$ वरील ब्रांच मागील इतिहासाची ब्रांच घेतल्यापासून घेते. जेव्हा वास्तविक अंदाजांची हार्डवेअर अंदाजांशी तुलना केली जाते, तेव्हा $a = 0$ वर फक्त एक प्रकरण अचूक असते आणि इतर सर्व प्रकरणे चुकीची असतात.

2-बिट डायनॅमिक ब्रांच प्रेडिक्शन

2 बिट डायनॅमिक ब्रांच प्रेडिक्शन 90% अचूकता देतात, कारण प्रेडिक्शन बिट बदलण्यापूर्वी प्रेडिक्शन दोनदा चुकीचा असणे आवश्यक आहे. 1-बिट ब्रांच प्रेडिक्शन एका चुकीच्या ब्रांच च्या प्रेडिक्शन नंतर, प्रेडिक्शन बिट उलट केला जातो. 2-बिट ब्रांच प्रेडिक्शन मध्ये दोन चुकीच्या प्रेडिक्शन नंतर, फक्त प्रेडिक्शन बिट उलट केला जातो.



स्कॅन करा

2-बिट डायनॅमिक ब्रांच प्रेडिक्शन समजून

2.8 रिस्क (RISC) पाईपलाईन

RISC हा एक इन्स्ट्रक्शन पाईपलाईन- रीडुस इन्स्ट्रक्शन सेट कॉम्पुटिंग आहे. इन्स्ट्रक्शन सेट ची साधेपणा काही उपक्रमांसह पाईपलाईनसाठी परवानगी देते जी प्रत्येक एक क्लॉक सायकल घेतात. रजिस्टर फिक्स्ड केले जात असताना फिक्स्ड लेन्थ इन्स्ट्रक्शन ची शैली डीकोडिंगला अनुमती देते. सर्व डेटा हाताळणी इन्स्ट्रक्शन रजिस्टर -ते- रजिस्टर तंत्राचा वापर करतात. सर्व ऑपरेंड रजिस्टरमध्ये असल्याने, एम्पेक्टिव्ह एंड्रेस किंवा मेमोरी रिट्रिव्हल ची आवश्यकता नाही. अशा प्रकारे, इन्स्ट्रक्शन पाईपलाईनमध्ये दोन किंवा तीन स्टेप्स असतात.

पहिला टप्पा मेमोरी तून इन्स्ट्रक्शन आणतो, आणि नंतर कार्यान्वित करतो. तिसरा टप्पा एएलयू ची गणना डेस्टिनेशन रजिस्टर मध्ये स्टोर करतात. RISC केवळ लोड आणि स्टोर इन्स्ट्रक्शन डेटा ट्रान्सफर करतात. या इन्स्ट्रक्शन अप्रत्यक्षपणे रजिस्टर करतात. तीन किंवा चार पाईपलाईन टप्पे वैशिष्ट्यपूर्ण आहेत. बहुतेक RISC प्रोसेसरमध्ये दोन मेमोरी असलेल्या दोन बस असतात-एक इन्स्ट्रक्शन साठी आणि एक डेटासाठी- मेमोरी ऍक्सेस कॉन्फ्लिक्ट कमी करण्यासाठी. इन्स्ट्रक्शन (I)-कॅश आणि डेटा (D)-कॅश CPU क्लॉक वर रन करतात. RISC प्रत्येक क्लॉक सायकल मध्ये एक इन्स्ट्रक्शन कार्यान्वित करते. CPU एकाच क्लॉक सायकल मध्ये प्रत्येक इन्स्ट्रक्शन कार्यान्वित करण्यासाठी पाईपलाईन केलेले आहे. RISC मध्ये पाईपलाईन विभाग आहेत ज्यात फक्त एका क्लॉक सायकल ची आवश्यकता असते, तर RISC मध्ये अनेक पाईपलाईन विभाग आहेत, ज्यापैकी सर्वात मोठे दोन किंवा अधिक क्लॉक सायकल घेतात.

2.9 वेक्टर प्रोसेसिंग

स्टॅंडर्ड संगणक काही संगणनाची कामे सोडवू शकत नाहीत. या समस्यांसाठी मोठ्या संख्येने गणनांची आवश्यकता असते, ज्या पूर्ण होण्यासाठी पारंपारिक संगणकाला अनेक दिवस किंवा आठवडेही लागू शकतात. विविध प्रकारच्या वैज्ञानिक आणि तांत्रिक समस्यांसाठी वेक्टर प्रक्रिया लागू आहे.

वेक्टर प्रोसेसिंग, हवामानाचा अंदाज, मानवी जीनोमचे मापिंग, पेट्रोलियम शोध, वैद्यकीय निदान, भूकंपाचा डेटा विश्लेषण, कृत्रिम बुद्धिमत्ता आणि तज्ञ प्रणाली, वायुगतिकी आणि अंतराळ उड्डाण अनुकरण यासारख्या विशेष अनुप्रयोगांमध्ये वेक्टर-प्रक्रिया संगणकांना मोठी मागणी आहे.

आधुनिक संगणकांशिवाय अनेक आवश्यक गणिते वाजवी कालावधीत पूर्ण केली जाऊ शकत नाहीत. उच्च गतीची इच्छित पातळी प्राप्त करण्यासाठी, वेक्टर आणि समांतर प्रक्रिया तंत्रांनी सुसज्ज सर्वात वेगवान आणि सर्वात विश्वासाहर् आधुनिक हार्डवेअर आवश्यक आहे.

अनेक उपयोजनांना मोठ्या अरे अरीथमेटिक कॉम्प्युटेशन ची आवश्यकता असते. फ्लोटिंगपॉइंट वेक्टर आणि माट्रिक्स हे पूर्णांक दर्शवतात. वेक्टर हे डेटा ऑब्जेक्ट्सचे एक आयामी अरे आहेत. V हा n लांबीचा रो वेक्टर ($V = [V_1, V_2, \dots, V_n]$) आहे. कॉलम वेक्टर हे कॉलममधील डेटा घटकांचे प्रतिनिधित्व करतात. माट्रिक्स गुणाकार ही वेक्टर प्रोसेसरची सर्वात संगणकीयदृष्ट्या गहन प्रक्रिया आहे. n^2 इनर प्रॉडक्ट्स किंवा n^3 मल्टिप्लाय-एंड ऑपरेशन्स दोन $n \times n$ माट्रिक्सचा गुणाकार करतात. $A \times m$ नंबर माट्रिक्स हा n रो वेक्टर किंवा m कॉलम वेक्टरचा सेट आहे.

2.10 अरे प्रोसेसर

अरे प्रोसेसर डेटाच्या मोठ्या अरेवर प्रक्रिया करू शकतो. या प्रकारच्या प्रोसेसरचे दोन प्रकारांमध्ये वर्गीकरण केले जाते: अटॅच अरे प्रोसेसर आणि SIMD (सिंगल इन्स्ट्रक्शन मल्टिपल डेटा) अरे प्रोसेसर. दोन्ही प्रकारचे अरे प्रोसेसर वेक्टरमध्ये फेरफार करू शकतात, जरी त्यांची रचना वेगळी असली तरी.



स्कॅन करा

वेक्टर ऑपरेशन्स आणि
मॅट्रिक्स गुणाकार समजून
घेणे

• अटॅच अँरे प्रोसेस:

सामान्य उद्देशाच्या संगणकाला संलग्न अँरे प्रोसेसर असतो. गुंतागुंतीच्या वैज्ञानिक अनुप्रयोगांसाठी वेक्टर प्रक्रिया संगणकाचा वेग वाढवते. अरीथमेटिक युनिटमध्ये पाईपलाईन केलेले फ्लोटिंग पॉईंट ऍडर्स आणि मल्टिप्लायर असतात.

होस्ट संगणक, एक सामान्य उद्देशाचा व्यावसायिक संगणक, जोडलेल्या प्रोसेसरद्वारे चालवला जातो. अँरे प्रोसेसर इनपुट-आउटपुट कंट्रोलरद्वारे संगणकाशी बाह्य इंटरफेस म्हणून काम करतो. वेगवान बस प्रोसेसर डेटा मुख्य मेमोरीमधून स्थानिक मेमोरीमध्ये हलवते. मूलभूत प्रोग्राम चालविताना सामान्य उद्देशाचा संगणक वापरकर्त्यांना पारंपारिक डेटा प्रक्रिया प्रदान करतो. जेव्हा गुंतागुंतीची अरीथमेटिक कार्ये अंमलात आणली जातात तेव्हा सोबतचा अँरे प्रोसेसर कार्य करतो. उदाहरणार्थ, VAX ची संगणकीय क्षमता 100 मेगाफ्लॉप पर्यंत वाढवण्यासाठी VAX-11 संगणक फ्लोटिंग-पॉईंट सिस्टीम्स FSP-164/MAX शी जोडला जातात.

• SIMD अँरे प्रोसेस:

वेक्टर डेटावर प्रक्रिया करण्यासाठी, SIMD अँरे प्रोसेसर अनेक कार्यात्मक युनिट्सचा वापर करतात. समांतर प्रक्रिया/कार्यात्मक युनिट उपस्थित आहेत.

या प्रक्रिया युनिट्सचा समन्वय एका कंट्रोल युनिटद्वारे एकत्र काम करण्यासाठी केला जातो जेणेकरून ते समान ध्येय साध्य करू शकतील. प्रत्येक प्रक्रिया घटकामध्ये (PE) स्वतःचे ALU, फ्लोटिंग-पॉईंट अरीथमेटिक युनिट (FPU) आणि स्वतःच्या स्थानिक मेमोरी M व्यतिरिक्त रजिस्टर असतात. मुख्य कंट्रोल युनिट हे PE व्यवस्थापनासाठी जबाबदार आहेत.

प्रोग्राम मुख्य मेमोरीमध्ये संग्रहित केला जातो. मास्टर कंट्रोल युनिट इंस्ट्रक्शनचे डीकोड करते आणि त्यांची अंमलबजावणी पद्धत ठरवीतात. मास्टर कंट्रोल युनिट थेट स्केलर आणि प्रोग्राम कंट्रोल इंस्ट्रक्शन कार्यान्वित करतात. वेक्टर इंस्ट्रक्शन एकाच वेळी सर्व PE ला पाठवल्या जातात. प्रत्येक PE ऑपरेंड साठविण्या स्थानिक मेमोरी चा वापर करतात. डारेक्टिवच्या समवर्ती अंमलबजावणीपूर्वी, वेक्टर ऑपरेंड स्थानिक मेमोरीमध्ये वितरित केले जातात.

एक उदाहरण म्हणून $C = A + B$ वेक्टर बेरीज विचारात घ्या. A आणि B चे i^{th} घटक प्रथम A आणि B मास्टर कंट्रोल युनिटद्वारे स्थानिक मेमोरी M मध्ये संग्रहित केले जातात, जेथे $i = 1, 2, 3, \dots, N$. सर्व PE ला फ्लोटिंग-पॉईंट अँड इंस्ट्रक्शन $c = a + b$ दिल्यामुळे एकाच वेळी जोडणी करणे शक्य आहे. C घटक प्रत्येक स्थानिक मेमोरीमध्ये नेहमी एकाच ठिकाणी साठवले जातात. वेक्टर बेरीज केवळ जोडणीच्या एका क्लॉक सायकल ने तयार केली जाऊ शकते. वेक्टर इंस्ट्रक्शनच्या अंमलबजावणीदरम्यान, PE परिस्थिती नियंत्रित करण्यासाठी मास्किंगचा वापर केला जातो. जेव्हा PE सक्रिय असते, तेव्हा त्याचे फ्लॅग सेट केले जातात आणि जेव्हा ते निष्क्रिय असते, तेव्हा ते रीसेट केले जातात. सहभागासाठी आवश्यक असलेले PE केवळ इंस्ट्रक्शन च्या अंमलबजावणीदरम्यान सक्रिय केले जातात.

ILLIAC IV संगणक हा इलिनॉय विद्यापीठात विकसित केलेला एक सुप्रसिद्ध SIMD अ‍ॅरि प्रोसेसर आहेत. हा संगणक आता निष्क्रिय आहेत. वेक्टर किंवा माट्रिक्स आधारित संख्यात्मक समस्यांसाठी SIMD प्रोसेसर अतिशय विशेष आहेत. तथापि, SIMD प्रोसेसर इतर मानक डेटा-प्रक्रिया अनुप्रयोगांसह अकार्यक्षम आहेत.

युनिट सारांश

- मायक्रोप्रोग्रॅम केलेले कंट्रोल युनिट संगणकातील मायक्रोऑपरेशन्सचे अनुक्रम सुरू करते. कार्यक्षमता हार्डवायर्ड किंवा मायक्रोप्रोग्रॅमिंग दृष्टिकोनातून अंमलात आणली जाऊ शकतात.
- हार्डवायर्ड दृष्टीकोन हार्डवेअरचा वापर करून कंट्रोल सिग्नल निर्माण करतो. तर, मायक्रोप्रोग्रॅमिंग आधारित कंट्रोल युनिट मायक्रोप्रोग्रॅम सेटयित करते जे मायक्रोइन्स्ट्रक्शन्सची मालिका तयार करते आणि सिग्नल नियंत्रित करतात.
- मायक्रोइन्स्ट्रक्शन्सच्या संग्रह कंट्रोल मेमोरी मध्ये स्टोर करून ठेवली जाते. प्रत्येक इन्स्ट्रक्शन चा स्वतःचा मायक्रोप्रोग्रॅम असतो जो कंट्रोल मेमोरी मध्ये विशिष्ट ठिकाणी साठवला जातो.
- कंट्रोल सिग्नल प्रोसेसरमधील रजिस्टर, अंतर्गत बस, ए. एल. यू. आणि विविध घटकांमधील मार्ग यासारखे विविध घटक सक्रिय करतात.
- अरीथमेटिक ऑपरेशन्स, i.e., बेरीज, वजाबाकी, गुणाकार आणि भागाकार अन साइन आणि साइन दोन्ही पूर्णांकांवर केली जाऊ शकतात.
- अपूर्णांक संख्या फिक्स्ड पॉइंट रेप्रेसेंटेशन आणि फ्लोटिंग पॉइंट रेप्रेसेंटेशन या दोन स्वरूपात दर्शविल्या जातात. दशांश पॉइंट फिक्स्ड पॉइंट रेप्रेसेंटेशन मध्ये हलत नाही.
- फ्लोटिंग पॉइंट रिप्रेझेंटेशनमध्ये, दशांश पॉइंट एकतर डावीकडे किंवा उजवीकडे जातो. दशांश पॉइंटच्या हालचालीवर आधारित, घातांक एकतर वाढवला किंवा कमी केला जातो.
- गणनेचा वेग वाढवण्यासाठी समांतरपणे विविध प्रकारच्या ऑपरेशन्सची गणना करण्यासाठी सुपरकंप्यूटर मध्ये अरीथमेटिक पाइपलाइन युनिट्सचा वापर केला जातो. उदाहरणार्थ, सुपर कॉम्प्युटर तीन एक्झिक्युशन युनिट्स वापरू शकतो, i.e., पूर्णांक संख्या अरीथमेटिक ऑपरेशन्स, लोड/स्टोअर ऑपरेशन्स, फ्लोटिंग पॉइंट नंबर अरीथमेटिक ऑपरेशन्स.
- इन्स्ट्रक्शन पाईपलाईन एकाच प्रोसेसर क्लॉक सायकल मध्ये अनेक इन्स्ट्रक्शन कार्यान्वित करू शकतात.
- पाईपलाईन इन्स्ट्रक्शन ची अंमलबजावणी हझार्ड च्या अनुपस्थितीत प्रणालीचे थ्रूपुट वाढवते. तथापि, इन्स्ट्रक्शन अंमलात आणल्या जात असताना संरचनात्मक, माहिती आणि कंट्रोल असे तीन भिन्न प्रकारचे पाईपलाईन हझार्ड उद्भवू शकतात आणि प्रणालीच्या कार्यक्षमतेमध्ये लक्षणीय बिघाड होऊ शकतात.
- हार्डवेअर रेसोर्स च्या मर्यादांमुळे संरचनात्मक हझार्ड उद्भवतात. हार्डवेअर रेसोर्स ची नक्कल करून या प्रकारचा हझार्ड दूर होऊ शकतात.
- इन्स्ट्रक्शन दरम्यान डेटा इंडिपेन्डन्स अस्तित्वात असल्यास डेटाचा हझार्ड असू शकतो. याचे RAW चे हझार्ड, WAR चे हझार्ड आणि WAW चे हझार्ड असे वर्गीकरण केले जाऊ शकतात.

- ब्रांच इन्स्ट्रक्शन मुळे कंट्रोल हार्डवर्ड उद्धवू शकतात. ब्रांच प्रेडिक्शन चा वापर करून कंट्रोल हार्डवर्ड कमी केले जाऊ शकतात. ब्रांच प्रेडिक्शन लावण्यासाठी दोन पद्धती आहेत: स्टॅटिक आणि डायनामिक ब्रांच प्रेडिक्शन.
- डायनामिक ब्रांच प्रेडिक्शन हे हिस्टरी बेस प्रेडिक्शन तंत्र आहे.
- RISC हा एक रेडूस इन्स्ट्रक्शन सेट कॉम्पुटिंग असलेला संगणक आहे जो कमीतकमी सब ऑपरेशन सह इन्स्ट्रक्शन पाइपलाइन तयार करू शकतो, ज्यापैकी प्रत्येक ऑपरेशन एका क्लॉक सायकल मध्ये केले जाते.
- वेक्टर आणि माट्रिक्सच्या संदर्भात वर्णन केलेल्या विज्ञान आणि अभियांत्रिकी समस्यांच्या उच्च गतीच्या गणनेसाठी वेक्टर-प्रक्रिया संगणक वापरले जातात.
- अरे प्रोसेसर डेटाच्या मोठ्या अरेवर प्रक्रिया करू शकतो. या प्रकारच्या प्रोसेसरचे दोन प्रकारांमध्ये वर्गीकरण केले जाते: संलग्न अरे प्रोसेसर आणि SIMD (सिंगल इन्स्ट्रक्शन मल्टिपल डेटा स्ट्रीम) अरे प्रोसेसर.

स्वाध्याय:**बहुपर्यायी प्रश्न**

प्रश्न. 2.1 कंट्रोल सिग्नल _____ कंट्रोल युनिट मध्ये संयुक्त तर्कशास्त्राद्वारे तयार केले जातात.

(अ) सूक्ष्म प्रोग्राम (ब) सॉफ्टवेअर (क) लॉजिक (ड) हार्डवायर्ड

प्रश्न. 2.2 सूक्ष्म इन्स्ट्रक्शन च्या संचाला _____ म्हणतात.

(अ) प्रोग्राम (ब) कमांड (क) मायक्रो प्रोग्राम (ड) मायक्रो कमांड

प्रश्न. 2.3 2 कॉम्पलिमेंट इन्स्ट्रक्शन मध्ये दर्शविल्या जाऊ शकणार्या संख्या म्हणजे

(अ) दोन्ही + ve आणि - ve संख्या (ब) 8 बिट्सपेक्षा मोठ्या संख्या (क) केवळ - ve संख्या (ड) केवळ + ve संख्या.

प्रश्न. 2.4 4-बिट 2 कॉम्पलिमेंट संख्येच्या प्रतिनिधित्वामध्ये, नकारात्मक व्हॅल्यूची श्रेणी

(अ) -1 ते-8 (ब) -1 ते-15 आहे. (क) - 1 ते-7 (ड) - 1 ते-16.

प्रश्न. 2.5 आधुनिक संगणकांमध्ये कोणती संख्या प्रणाली लोकप्रिय आहे?

(अ) साइन आणि माग्रीटूड (ब) ऑक्टल (क) 1 कॉम्पलिमेंट (ड) 2 कॉम्पलिमेंट.

प्रश्न. 2.6 0-1 बायनरी सबट्रॅक्शनसाठी डिफरन्स आणि बॉरोव्ह व्हॅल्यू काय आहेत?

(अ) 0,1 (ब) 0,0 (क) 1,1 (ड) 1,0.

प्रश्न. 2.7 0110-0010 एक बायनरी संख्या दुसऱ्या बायनरी संख्या पासून वजा केल्यास काय होते?

(अ) 1000 (ब) 0100 (क) 0110 (ड) 0011

प्रश्न. 2.8 2 कॉम्पलिमेंट संख्या प्रणालीमध्ये $(-87)_{10}$ कसे दर्शविले जाते?

(अ) 11011011 (ब) 10101001 (क) 11010110 (ड) 10110011.

प्रश्न. 2.9 इन्स्ट्रक्शन पाइपलाइनमध्ये _____ नाही?

(अ) एंड्रेस हझार्ड (ब) कंट्रोल हझार्ड (क) डेटा हझार्ड (ड) संरचनात्मक हझार्ड.

प्रश्न. 2.10 जर इन्स्ट्रक्शन X ने इन्स्ट्रक्शन (X-1) द्वारे लिहिण्यापूर्वी काही डेटा सुधारित करण्याचा प्रयत्न केला तर त्याचा _____ हझार्ड होतो.

(अ) RAR (ब) RAW (क) WAR (ड) WAW.

प्रश्न. 2.11 पाच इन्स्ट्रक्शन आहेत, ज्या खाली दिल्या आहेत.

I1: Mul R10, R11, R12

I2: Add R15, R10, R12

I3: Add R13, R10, R14

I4: Mul R12, R11, R13

I5: Sub R15, R16, R17

खालील तीन विधाने विचारात घ्या:

1: I2 आणि I5 इन्स्ट्रक्शन मध्ये अँटी डिपेन्डन्स आहे.

2: I2 आणि I4 इन्स्ट्रक्शन मध्ये अँटी डिपेन्डन्स आहे.

3: अँटी डिपेन्डन्स वृत्ती नेहमी इन्स्ट्रक्शन पाइपलाईनमध्ये एक किंवा अधिक स्टॉल्स तयार करते.

वरीलपैकी कोणती विधाने बरोबर/बरोबर आहेत?

(अ) विधान 1 सत्य आहे

(ब) विधान 2 सत्य आहे

(क) विधान

1 आणि 3 सत्य आहेत

(ड) विधान 2 आणि 3 सत्य आहेत.

प्रश्न. 2.12 खालील इन्स्ट्रक्शन चा विचार करा:

I1 : Add R10, R7, R12 ;

$R10 \leftarrow R7 + R12$

I2 : Sub R12, R7, R10 ;

$R12 \leftarrow R7 - R10$

I3 : Div R12, R9, R12 ;

$R12 \leftarrow R9 / R12.$

या इन्स्ट्रक्शन दरम्यान अस्तित्वात असलेल्या डेटा हार्डवर्ड चे प्रकार दाखवा.

(अ) RAW, WAR, WAW (ब) RAW, WAR (क) RAW, WAW (ड) WAR, WAW

प्रश्न. 2.13 जेव्हा 100010010 ला 1101 ने भागले जाते, तेव्हा काय उर्वरित राहते

(अ) 1

(ब) 0

(क) 11

(ड) 101

प्रश्न. 2.14 पाइपलाईन प्रोसेसरसाठी 2 डिझाईन्स आहेत. डिझाइन-1 मध्ये 3ns, 2 ns, 4 ns, 2ns आणि 3 ns च्या अंमलबजावणीच्या वेळेसह 5 स्टेजेस ची पाइपलाईन आहे. तर डिझाइन-2 मध्ये 8 पाइपलाईन स्टेजेस आहेत ज्यात प्रत्येकी 2 ns अंमलबजावणी वेळ आहे. 100 इन्स्ट्रक्शन अंमलात आणण्यासाठी डिझाइन डी 2 ओव्हर डिझाइन डी 1 वापरून किती वेळ वाचवता येईल?

[संकेत: पाइपलाईनसाठी एक्झिक्युशन वेळ = $(K + n - 1) * \text{एका टप्प्यातील एक्झिक्युशन वेळ}$; जेथे k = पाइपलाईनमधील स्टेजेस ची संख्या, n = इन्स्ट्रक्शन ची संख्या, एका टप्प्याची अंमलबजावणी वेळ = कमाल (सर्व टप्प्यांची अंमलबजावणी वेळ)].

(अ) 314 ns

(ब) 202 ns

(क) 96 ns

(ड) 190 ns.

प्रश्न. 2.15 7-बिट शब्दांमध्ये, एक यंत्र फ्लोटिंग पॉइंट नंबर सेटयित करते. पहिला बिट क्रमांकाचे साइन दर्शवितो, पुढील तीन बेस्ड एक्सपोनन्ट दर्शवतात आणि शेवटचे तीन मान्तिताच्या परिमाण दर्शवतात. वरील वर्डत, आपण 33.35 प्रतिनिधित्व करणे आवश्यक आहे. या प्रकरणात, त्रुटी असेल

अ) ओव्हरफ्लो

(ब) अंडरफ्लो

(क) एरर

(ड) एरर नाही.

बहुपर्यायी प्रश्नाचे उत्तरे

2.1	(ड)	2.2	(क)	2.3	(अ)	2.4	(अ)	2.5	(ड)	2.6	(क)
2.7	(ब)	2.8	(ब)	2.9	(अ)	2.10	(ड)	2.11	(अ)	2.12	(अ)
2.13	(अ)	2.14	(अ)	2.15	(अ)						

लघु आणि दीर्घ उत्तराचे प्रश्न:**श्रेणी-I**

प्रश्न. 2.1 प्रोसेसर कंट्रोल युनिट कार्य कसे करतात?

प्रश्न. 2.2 कंट्रोल मेमोरी ची भूमिका काय आहे?

प्रश्न. 2.3 हॉरीझॉन्टल आणि व्हर्टिकल मायक्रोप्रोग्रॅम कंट्रोल युनिटची वैशिष्ट्ये वर्णन करा.

प्रश्न. 2.4 मायक्रोप्रोग्रॅम कंट्रोल युनिटची प्राथमिक कार्ये कोणती आहेत?

प्रश्न. 2.5 काही सामान्य मायक्रोप्रोग्रॅमिंग अनुप्रयोगांचे वर्णन करा.

प्रश्न. 2.6 इन्स्ट्रक्शन आणि मायक्रोऑपरेशन्स एकमेकांशी कसे संबंधित आहेत?

प्रश्न. 2.7 कंट्रोल युनिट इनपुट आणि आउटपुटचे वर्णन करा.

प्रश्न. 2.8 कंट्रोल ह्र्झार्ड आणि डेटा ह्र्झार्ड यांच्यातील फरक स्पष्ट करा.

प्रश्न. 2.9 फ्लोटिंग पॉइंट नंबरचे चार मुख्य घटक कोणते आहेत?

प्रश्न. 2.10 फ्लोटिंग पॉइंट नंबरच्या घातांक भागासाठी बाईज्ड प्रतिनिधित्व का आवश्यक आहे?

प्रश्न. 2.11 साइन आणि माग्नीटूड आणि 2 कॉम्पलिमेन्ट प्रतिनिधित्व वापरून नकारात्मक संख्या कशी ठरवायची.

प्रश्न. 2.12 फ्लोटिंग पॉइंट नंबरसाठी तुम्ही बेरीज आणि वजाबाकी कशी करता?

प्रश्न. 2.13 वेक्टर प्रोसेसिंग आणि अरे प्रोसेसरमध्ये काय फरक आहे?

प्रश्न. 2.14 संरचनात्मक धोक्याचे आणि त्याच्या समाधानाचे उदाहरण द्या.

प्रश्न. 2.15 स्टॅटिक आणि डायनॅमिक ब्रँच प्रेडिक्शनमध्ये काय फरक आहे?

श्रेणी-II

प्रश्न. 2.16 हार्डवायर्ड आणि मायक्रोप्रोग्रॅम कंट्रोल युनिट अंमलबजावणी स्पष्ट करा. कंट्रोल मेमोरीशी संबंधित हार्डवायर्ड कंट्रोल युनिट सोबत असोसिएशन व्यवहार्य आहे का?

प्रश्न. 2.17 कंट्रोल युनिटमध्ये, एड्रेस सिक्वेन्सिंग कसे कार्य करतात? "मायक्रोऑपरेशन", "मायक्रोइन्स्ट्रक्शन" आणि "मायक्रोप्रोग्रॅम" हे वर्ड कंट्रोल युनिटमध्ये वापरले जातात ते परिभाषित करा.

प्रश्न. 2.18 पाच स्टेज ची पाईपलाईन असलेल्या संगणकात खालील इन्स्ट्रक्शन अंमलात आणल्या जातात याचा विचार करा:

Mul R9, R8, #20

Add R11, R10, #3

Or R12, R10, #100

Subtract R13, R8, R10

पाइपलाईनच्या प्रत्येक स्टेज साठी एक क्लॉक सायकल आवश्यक आहे. पाइपलाईन आकृतीद्वारे इन्स्ट्रक्शन च्या अंमलबजावणीचे प्रतिनिधित्व करा. पाईपलाईनचा प्रत्येक स्टेज किती कार्य करतो त्याचे वर्णन करा.

प्रश्न. 2.19 इन्स्ट्रक्शन पाईपलाईन आणि अरीथमेटिक पाईपलाईन मधील फरक स्पष्ट करा. संरचनात्मक हार्डवेअर, इन्स्ट्रक्शन हार्डवेअर आणि कंट्रोल हार्डवेअर यांची पाईपलाईनमध्ये चर्चा करा. या हार्डवेअर चा पाईपलाईनच्या कार्यक्षमतेवर परिणाम का होतो? या हार्डवेअर ना तोंड देण्यासाठी उपाययोजनांची चर्चा करा.

प्रश्न. 2.20 वेक्टर प्रोसेसर अनुप्रयोगांची चर्चा करा. SIMD ऑपरेशन प्रोसेसरचे सर्किट तयार करा आणि ते कसे कार्य करते याचे वर्णन करा.

संख्यात्मक प्रॉब्लम:

प्रश्न. 2.21 तुम्ही 16 बिट्स वापरून साइन आणि माग्नीट्यूड, 2 कॉम्पलिमेंट 512 आणि -29 दशांश संख्या प्रतिनिधित्व कसे दर्शवाल?

[उत्तर: साइन आणि माग्नीट्यूड: 0000.0010.0000.0000 (512) 1000.0000.0001.1101 (-29)

2 कॉम्पलिमेंट प्रतिनिधित्व: 0000.0010.0000 (512) 1111.1111.1110.0011 (-29)].

प्रश्न. 2.22 तुम्ही 2 कॉम्पलिमेंट बायनरी व्हॅल्यू 1101011 आणि 0101101 दशांश मध्ये कशी दर्शवाल?

[उत्तर: -21; 45].

प्रश्न. 2.23 क्रमाने 100 कामे करण्यासाठी प्रत्येक विभागात आवश्यक असलेल्या 20 ns च्या क्लॉक सायकल सह 4- स्टेज पाईपलाईन प्रणाली गृहीत धरा. स्पीडअप रेशो किती आहे?

[उत्तर: 3.88]

प्रश्न. 2.24 चला कल्पना करूया की 2.5 गीगाहर्ट्झ, नॉन- पाईपलाईन प्रोसेसर प्रति इन्स्ट्रक्शन सरासरी 4 सायकल लागतात. त्याच प्रोसेसरमध्ये पाच स्टेज ची पाईपलाईन सादर केली जाते, परंतु पाईपलाईनच्या अंतर्गत विलंबामुळे क्लॉक ची स्पीड 2 गीगाहर्ट्झपर्यंत खाली येते. गृहीत धरा की पाईपलाईन विलंब न करता सुरळीतपणे पुढे जात आहे. ही पाईपलाईन प्रोसेसर किती वेगवान आहे?

[उत्तर: 3.2]

प्रश्न. 2.25 चार स्टेज च्या पाईपलाईन प्रोसेसरचा विचार करा. खालील तक्त्यात S1, S2, S3 आणि S4 टप्प्यांमध्ये I1, I2, I3 आणि I4 या चार इन्स्ट्रक्शन साठी आवश्यक असलेल्या सायकल ची संख्या दर्शविली आहे.

	S1	S2	S3	S4
I1	2	1	1	1
I2	1	3	2	2
I3	2	1	1	3
I4	1	2	2	2

पुढील लूप पूर्ण करण्यासाठी किती सायकल ची आवश्यकता आहे?

```
for(i=1 to 2) { I1; I2; I3; I4; }
```

[उत्तर: 23]

प्रश्न. 2.26 1-बिट ब्रांच प्रेडिक्टरसह दिलेल्या लूप साठी किती चुकीचे प्रेडिक्शन आहेत?

```
for (i=0; i<5; i++)
```

```
{
```

```
a+=5;
```

```
}
```

सुरुवातीला घेतलेल्या ब्रांच मधून प्रेडिक्शन सुरु करण्याचे गृहीत धरा आणि तुम्ही ही इट्रेशन दोनदा चालवाल.

[उत्तर: 3]

प्रश्न. 2.27 $A = 1000$ आणि $B = 0011$ या अन साइन संख्यांवर नॉन रिस्टोरिंग डिव्हिजन पद्धत वापरून $A \div B$ करा. भाग आणि शेष यांची अंतिम व्हॅल्यू काय असतील?

[उत्तर: भाग = 0010, शेष = 00010]

प्रश्न. 2.28 2 कॉम्प्लिमेंट संकेतात $x = 0101$ आणि $y = 1010$ संख्या दिल्यास, गुणाकार $p = x * y$ मिळविण्यासाठी बूथचा अल्गोरिदम वापरा.

[उत्तर: 1110 0010]

प्रश्न. 2.29 तुम्ही ही संख्या IEEE 32-बिट फ्लोटिंग-पॉइंट स्वरूपात कशी लिहाल:

(a)-5 (b)-1.5 (c) 384 (d)-1/32

[उत्तर: (a) 1 10000001 010000000000000000000000]

(b) 1 01111111 100000000000000000000000

(c) 0 10001111 100000000000000000000000

(d) 1 01111010 000000000000000000000000]

प्रश्न. 2.30 8-बिट बायस्ड एक्स्पोनंट आणि 23 बिटच्या मान्टिसासह फ्लोटिंग-पॉइंट स्वरूपाचा विचार करा. या स्वरूपात, -720 आणि 0.645 संख्यांसाठी बिट नमुना दर्शवा.

[उत्तर: (a) 1 10001000 011010000000000000000000;

(b) 0 01111010 01001010000111101100000]



स्कॅन करा

नवशिक्यांसाठी Xilinx

व्हिड्योरिअल

प्रात्याशिक PRACTICAL

उद्देश: व्हेरिलॉग हार्डवेअर वर्णन लॅंग्वेज वापर करून बेरीज, वजाबाकी, बूथचा गुणाकार आणि 2 कॉम्पलिमेन्ट संख्यांसाठी विभाजन पुनर्संचयित करणारे हार्डवेअर सर्किट तयार करा.

साधने: Xilinx ISE डिझाइन स्वीट [5]

सिद्धांत: व्हेरिलॉगमध्ये मूलभूत प्रोग्रॅम कसा तयार करावा हे आधी अध्याय 1 मध्ये समाविष्ट केले आहे (practical). 2 कॉम्पलिमेन्ट संख्यांसाठी संगणकाच्या अरीथमेटिक च्या अल्गोरिदमची चर्चा यापूर्वी अध्याय 1 मध्ये केली गेली आहे.

हे अल्गोरिदम व्हेरिलॉग [5] वापरून Xilinx ISE डिझाइन सूट टूलमध्ये अंमलात आणले जाऊ शकतात. नवशिक्यांसाठी व्हेरिलॉग ट्यूटोरियलचा सल्ला घेऊन तुम्ही प्रगत व्हेरिलॉग मॉड्यूल लेखन देखील शिकू शकता. मल्टीप्लेक्सर्स, फ्लिपफ्लॉप्स आणि विविध फंक्शनल युनिट्सचे कॉम्बिनेशन सर्किट यासारखे असंख्य बिल्डिंग ब्लॉक्स कसे तयार करावे हे तुम्ही शिकू शकता.

2 कॉम्पलिमेन्ट संख्यांचा वापर करून खालील ऑपरेशन्स कार्यान्वित करण्यासाठी आवश्यक इनपुट आणि आउटपुट रजिस्टरची संख्या, तसेच कॉम्बिनेशन सर्किटचे वर्णन या अध्यायात केले आहे.

- 1) बेरीज
- 2) वजाबाकी
- 3) बूथचा गुणाकार
- 4) रिस्टोरिंग डिव्हिजन

प्रक्रिया:

1. प्रथम, सर्किटच्या इनपुट आणि आउटपुटची संख्या पहा.
2. मुख्य मॉड्यूल लिहा

module signed_arithmetic (list of input and output ports separated by comma)

(verilog program here)

endmodule

3. दोन इनपुट व्हेरिएबल्सची नावे घोषित करा ज्यात दोन साइन संख्या साठवल्या जातील.
4. आउटपुट पोर्टची नावे घोषित करा
5. एका वायरला इंटरमिजिएट आउटपुट म्हणून नियुक्त करा जे दुसऱ्या सर्किटचे इनपुट बनेल.



स्कॅन करा
नवशिक्यांसाठी व्हेरिलॉग
ट्यूटोरियल



स्कॅन करा
व्हेरिलॉगमध्ये बूथचा
गुणाकार प्रोग्रॅम



स्कॅन करा
xilinx मध्ये बूथ गुणाकार चे
सिम्युलेशन

6. बेरीज, वजाबाकी, बूथ गुणाकार आणि रिस्टोरिंग डिव्हिजन चयित करण्यासाठी चार मूलभूत उदाहरणांची यादी.

7. सर्किटचे प्रत्येक उदाहरण एका नावाने सुरू होते, जसे की बेरीज, वजाबाकी, बूथ _ मल्टिप्लिकेशन, रिस्टोरिंग _ डिव्हिजन त्यानंतर इन्स्टन्स आउटपुट आणि अल्पविरामाद्वारे वेगळे केलेले आणि कंसात संलग्न केलेले इनपुट.

8. इनपुटला दोन साइन संख्या केलेले क्रमांक देऊन रचनेची पडताळणी करा. परिपथ पडताळणीसाठी चाचणीपीठ तयार करा. टेस्टबेंचमध्ये सर्व संभाव्य इनपुट आणि आउटपुट संयोजनांचा उल्लेख करा.

9. आउटपुट पाहण्यासाठी आणि सर्किटचे कार्य तपासण्यासाठी Xilinx मध्ये Isim चालवा. जर आउटपुट सर्व संभाव्य संयोगांसह बरोबर असेल, i.e., दोन सकारात्मक संख्या, एक सकारात्मक आणि एक नकारात्मक, दोन्ही नकारात्मक संख्या नंतर सर्किट योग्यरित्या डिझाइन केले आहे.

अधिक जाणून घ्या:

पेंटियम चिपचे जनक विनोद धाम यांचा जन्म 1950 च्या दशकात भारतातील पुणे येथे झाला. धाम यांनी वयाच्या 21 व्या वर्षी 1971 मध्ये दिल्ली कॉलेज ऑफ इंजिनिअरिंगमधून इलेक्ट्रिकल इंजिनिअरिंगमध्ये अभियांत्रिकी पदवी प्राप्त केली. अभियांत्रिकीची पदवी पूर्ण केल्यानंतर, त्यांनी कॉन्टिनेंटल डिव्हाइसेसमध्ये चार वर्षे काम केले, त्या वेळी भारतातील मोजक्या खाजगी सिलिकॉन सेमीकंडक्टर स्टार्ट-अप्सपैकी एक, ज्याने अमेरिकन टेराडाइन सेमीकंडक्टर कंपनीशी भागीदारी केली, जिथे त्यांना सेमीकंडक्टर्समध्ये रस निर्माण झाला. या उद्योगात यशस्वी होण्यासाठी, त्यांचा असा विश्वास होता की भौतिकशास्त्राची अधिक चांगली समज सेमीकंडक्टर उपकरणांच्या वर्तनास चालना देते.

1975 मध्ये त्यांनी ओहायो येथील सिनसिनाटी विद्यापीठात भौतिकशास्त्रात पदव्युत्तर पदवी घेण्यासाठी प्रवेश घेतला. 1977 मध्ये मास्टर ऑफ सायन्स प्राप्त केल्यानंतर, त्यांनी डेटन, ओहायो येथील एन. सी. आर. कॉर्पमध्ये अभियंता म्हणून काम करण्यास सुरुवात केली, जिथे त्यांनी प्रगत अस्थिर नसलेली मेमोरी विकसित केली. अस्थिर नसलेल्या आठवणींवर त्यांनी केलेल्या अग्रगण्य कार्यामुळे मिश्र अस्तरविद्युत तंत्र आणि अस्थिर नसलेल्या मेमोरी उपकरणासाठी एन. सी. आर. च्या 1985 च्या पेटंटसाठी मार्ग मोकळा झाला. नंतर ते इंटेल कॉर्पोरेशनचे अभियंता झाले. पेंटियम मायक्रोप्रोसेसरच्या निर्मितीतील त्यांच्या योगदानामुळे त्यांना 'पेंटियम अभियंता' ही उपाधी मिळाली आहे. याव्यतिरिक्त, तो इंटेलमधील फ्लॅश मेमोरी टेक्नॉलॉजीच्या मूळ निर्मात्यांपैकी एक आहे. इंटेलच्या मायक्रोप्रोसेसर समूहाचे उपाध्यक्ष म्हणून त्यांची नियुक्ती करण्यात आली.

आयुर्वेदाचा इतिहास

आयुर्वेद हे दोन शब्दांचे संयोजन आहे: आयु म्हणजे जीवन आणि वेद म्हणजे शहाणपण. ऋग्वेद, सामवेद, यजुर्वेद आणि अथर्ववेद हे भारतीय औषधोपचार आणि निरोगी जीवनाचा पाया होते. आयुर्वेद



हे अथर्ववेदाशी संबंधित असलेल्या उपवेदांपैकी एक आहे हे सर्वश्रुत आहे.

अथर्ववेद हा जादूच्या मंत्रांचा आणि गूढ शाखांचा संग्रह आहे, तसेच आयुर्वेदाची वैद्यकीय पद्धत आहे, जी आजार, दुखापती, वंध्यत्व, विवेक आणि एकूण आरोग्य बरे करण्यासाठी वापरली जाते. आयुर्वेद सर्व जीवनशैली हाताळते. परिणामी, रुग्णांना योग, अरोमाथेरापी, ध्यान, रत्ने, ताबीज, औषधी वनस्पती, आहार, ज्योतिष, रंग आणि शस्त्रक्रियेसह त्यांच्या आरोग्याच्या सर्व पैलूंचा विचार करून उपचार दिले जातात. मर्म हे त्यांच्या संवेदनशीलतेसाठी ओळखले जाणारे शरीराचे भाग आहेत आणि आयुर्वेद त्यांना संबोधित करते. योग, मालिश आणि व्यायामाच्या इतर प्रकारांची शिफारस केली जाते.

चारक संहिता म्हणून ओळखला जाणारा संस्कृत कवितांचा संग्रह चारकाने पहिल्या शतकात (A.D.) संकलित केला होता. सुश्रुत आणि वागभट हे दोघेही पुस्तकांचे लेखक होते. सुश्रुत संहिता बहुधा चौथ्या शतकात (A.D.) रचली गेली असावी. अष्टांग हृदय आणि संग्राम हे तिसरे सर्वात महत्वाचे ग्रंथ वाघवताने पाचव्या शतकात (A.D.) लिहिले होते. चरक आणि सुश्रुत यांनी स्थापन केलेल्या औषधी आणि शस्त्रक्रियेच्या पद्धती आयुर्वेदाची मुळे मानल्या जातात.

धन्वंतरी भवप्रकाश, राजा आणि शालीग्राम यासह निघंतुस म्हणून ओळखल्या जाणार्या सोळा महत्त्वपूर्ण पूरकांच्या जोडणीद्वारे आयुर्वेद पुढे विकसित केले गेले. नवीन औषधांनी जुन्या औषधांची जागा घेतली आहे जी अयशस्वी झाली होती. असे वाटत होते की याची उपयुक्तता वाढत आहे, नवीन आजार शोधले जात आहेत आणि इतर उपचार शोधले जात आहेत. या आहारातील पूरकांमध्ये सुमारे दोन हजार वेगवेगळ्या औषधी वनस्पतींचा समावेश होता [7,8].

संदर्भ आणि सुचविलेले वाचन

[1] एम. मॉरिस मनो, संगणक प्रणाली वास्तुकला. प्रेंटिस-हॉल, इंक., तिसरी आवृत्ती.

<https://poojavaishnav.files.wordpress.com/2015/05/mano-m-m-computer-systemarchitecture.pdf> (last accessed: Oct 2022)

[2] कार्ल हमाकर, इवोन्को व्रॅनेसिक, सफवत झाकी आणि नरैग मंजिकियन, संगणक संस्था आणि एम्बेडेड सिस्टीम्स. माकग्रा-हिल उच्च शिक्षण, 2011.

[3] विल्यम स्टॅलिंग, कॉम्प्युटर ऑर्गनायझेशन अँड आर्किटेक्चर डिझायनिंग फॉर परफॉर्मन्स. 10वी आवृत्ती, 2016.

इन्स्टिट्यूट ऑफ इलेक्ट्रिकल अँड इलेक्ट्रॉनिक्स इंजिनियर्स, आयईईई स्टँडर्ड फॉर बायनरी फ्लोटिंग-पॉइंट एरिथमेटिक, एएनएसआय/आयईईई स्टँडर्ड 754-2008, ऑगस्ट 2008.

[5] Xilinx ISE डिझाइन सेट.

<https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive-ise.html> (last accessed: Oct 2022)

[6] विनोद धाम यांचे चरित्र. <http://www.thinklink.in/blog/father-of-the-pentium-chip-vinoddham> (last accessed: Oct 2022)

[7] आयुर्वेद: एक विहंगावलोकन. [https://yehaindia.com/ayurveda-an-overview /](https://yehaindia.com/ayurveda-an-overview/) (last accessed: Oct 2022)

[8] आयुर्वेदाचा इतिहास... उपचारांचा वारसा. <https://poliklinika-harni.hr/images/uploads/434/povijest-ayurvede.pdf> (last accessed: Oct 2022)

[9] डॉ. जॉन जोस यांचा एनपीटीईएल अभ्यासक्रम, प्रगत संगणक वास्तुकला, आयआयटी गुवाहाटी, 2019. [https://archive.nptel.ac.in/cours/106/103/106103206 /](https://archive.nptel.ac.in/cours/106/103/106103206/) (last accessed: Oct 2022)

[10] प्रा. इंद्रनील सेनगुप्ता आणि प्रा. कमलिका दत्ता यांचा एनपीटीईएल अभ्यासक्रम, संगणक वास्तुकला आणि संघटना, आयआयटी खरगपूर, 2017. [https://archive.nptel.ac.in/cours/106/105/106105163 /](https://archive.nptel.ac.in/cours/106/105/106105163/) (last accessed: Oct 2022)

3. मायक्रोप्रोसेसर आर्किटेक्चर

युनिटची वैशिष्ट्ये

या युनिट मध्ये खालील पैलूंवर चर्चा केली आहे:

- मायक्रोप्रोसेसरची मूलभूत तत्त्वे;
- इन्स्ट्रक्शन सेट आर्किटेक्चर;
- प्रोग्रामरच्या दृष्टीकोनातून रचना तत्त्वे;
- इंटेल 8086 मायक्रोप्रोसेसर ची केस स्टडी;

अधिक जिज्ञासा आणि सर्जनशीलता वाढवण्यासाठी आणि समस्या सोडवण्याची कौशल्ये वाढवण्याच्या उद्देशाने विषयांचे व्यावहारिक अनुप्रयोग सादर केले जातात. बहुपर्यायी प्रश्नांच्या मोठ्या संख्येव्यतिरिक्त, ब्लूमच्या वर्गीकरण, अर्थात: लहान आणि दीर्घ उत्तर-प्रश्नांच्या खालच्या किन्वा उच्च स्तरांनुसार पुढील दोन श्रेणीचे प्रश्न तयार केले जातात. युनिट संख्यात्मक समस्या स्वरूपात सराव असाइनमेंट, संदर्भांची यादी आणि सुचविलेले वाचन प्रदान करते. हे देखील रजिस्टरवले गेले आहे की आवश्यक पूरक साहित्य मिळविण्यासाठी अनेक क्यू. आर. कोड विविध विभागांमध्ये समाविष्ट केले गेले आहेत.

सामग्रीवर आधारित संबंधित प्रात्यक्षिक विषयावरील "अधिक जाणून घ्या" विभाग आहे. हा विभाग काळजीपूर्वक अशा प्रकारे तयार करण्यात आला आहे की त्यात असलेली पूरक माहिती पुस्तकाच्या वाचकांसाठी मौल्यवान असेल. हा विभाग प्रामुख्याने संगणक प्रणाली संघटनेच्या विकासासाठी भारतीय नवसंशोधकांच्या योगदानावर लक्ष केंद्रित करतो, अन्नपदार्थांमध्ये आढळणारी A ब्लॉक सूक्ष्म पोषक तत्त्वे भारतीय मातीची भांडी शिजवतात आणि निरोगी राहण्यासाठी पौष्टिक अन्न तयार करण्यात ही महत्त्वाची भूमिका बजावते.

तर्क

या अध्यायात स्टॅंडर्ड 8-बिट आणि 16-बिट मायक्रोप्रोसेसर आर्किटेक्चरवर चर्चा केली आहे. या मायक्रोप्रोसेसरची इन्स्ट्रक्शन सेट आर्किटेक्चर RISC किंवा CISC आहे. CISC इन्स्ट्रक्शन संचामध्ये परिवर्तनीय लांबी आणि अधिक अत्याधुनिक आदेश असतात. दुसरीकडे, RISC फिक्स्ड-लांबीच्या इन्स्ट्रक्शन वापरते जे कमी कठीण असतात. RISC डेटा रजिस्टरमध्ये संग्रहित करतात आणि लोड आणि स्टोअर इन्स्ट्रक्शनद्वारे मेमोरीमध्ये मर्यादित प्रवेश आहे. सोयीसाठी, प्रोग्रामर कार्यक्षम पद्धतीने प्रोग्राम लिहिण्यासाठी अनेक इन्स्ट्रक्शन स्वरूप आणि

अड्रेसिंग मोड वापरू शकतात. या अध्यायात 8086 मायक्रोप्रोसेसरच्या संरचनेचे सखोल वर्णन केले गेले आहे. मायक्रोप्रोसेसर एक्झिक्युशन युनिट हे अनेक फ्लॅग, रजिस्टर आणि विशेष उद्देश कंट्रोल फ्लॅग नी बनलेले असते. बस इंटरफेस युनिटमध्ये सेगमेंट रजिस्टर, एक इन्स्ट्रक्शन क्यू, एक कंट्रोल युनिट आणि ऍड्रेस कॅल्कुलेशन प्रक्रिया युनिट असते.

पूर्व-आवश्यकता :

संगणक प्रणाली संघटना (Unit-I)

असेम्ब्ली लॅंग्वेज प्रोग्रामिंग: मूलभूत इन्स्ट्रक्शन चे मूलभूत ज्ञान (पॉलिटेक्निक अभियांत्रिकी)

युनिटचे निष्पत्ती (UNIT OUTCOMES):

या युनिटच्या निष्पत्तीची यादी खालीलप्रमाणे आहे.

U3-O1: सिस्टिम डिझाइन मायक्रोप्रोसेसरच्या भूमिकेचे वर्णन करणे.

U3-O2: इन्स्ट्रक्शन सेट आर्किटेक्चरचे वर्णन करणे.

U3-O3: प्रोग्रामरच्या दृष्टीकोनातून रचना तत्त्वे.

U3-O4: इंटेल 8086 मायक्रोप्रोसेसरची वैशिष्ट्ये स्पष्ट करणे.

युनिट -3 निष्पत्ती	निष्पत्ती सह अपेक्षित मापिंग				
	2- कमकुवत मापिंग, 2- मध्यम मापिंग, 3- मजबूत मापिंग				
	CO-1	CO-2	CO-3	CO-4	CO-5
U3-01	3	3	3	3	2
U3-02	2	2	3	3	2
U3-03	2	2	3	2	2
U3-04	3	2	2	1	1

3.1 परिचय

मायक्रोप्रोसेसर ही ALU (एरिथमेटिक लॉजिक युनिट) रजिस्टर आणि कंट्रोल सर्किटरी असलेली सिलिकॉन चिप आहे. सेंट्रल प्रोसेसिंग युनिट (सीपीयू) मध्ये एरिथमेटिक लॉजिक कार्ये करण्यासाठी एरिथमेटिक लॉजिक युनिट, मध्यवर्ती व्हॅल्यू संचयीत करण्यासाठी रजिस्टर बँक, ज्यातून सीपीयू अनेकदा ऑपरेंड ओढतात आणि संपूर्ण प्रोसीजरचे नियमन करणारे कंट्रोल लॉजिक असते. पूर्वी, सीपीयू आर्किटेक्चर हे एएलयू आर्किटेक्चर, स्वतंत्र रजिस्टर सर्किट्स आणि स्वतंत्र कंट्रोल लॉजिक डिझाइन यासारख्या वेगळ्या स्वतंत्र घटकांमध्ये विभागले गेले होते.

वेगवेगळ्या मॉड्युल्सची समस्या अशी आहे की त्यांच्या चिप्स भीन्न- भीन्न असतात. जेव्हा हे सर्व घटक प्रिन्टेड सर्किट बोर्डमध्ये समाविष्ट केले जातात, तेव्हा ते तांब्याच्या तारांद्वारे जोडले जातात. बाह्य लाइनच्या वेगामुळे प्रणालीचा वेग मर्यादित असतो; या रचनेसह खूप उच्च वेग प्राप्त केला जाऊ शकत नाही. त्यामुळे, हाय-स्पीड आर्किटेक्चरसाठी, हे सर्व CPU घटक एकाच चिपवर असणे आवश्यक आहे.

मायक्रोप्रोसेसर ही संज्ञा 'मायक्रो' आणि 'प्रोसेसर' या शब्दांचे मिश्रण आहे. प्रोसेसर हे डेटा, विशेषतः बायनरी पूर्णांकांच्या प्रोसीजरसाठी एक उपकरण आहे. यावर काही प्रक्रिया केली जातात. मायक्रोप्रोसेसरसाठी डिझाइन टीम संख्यांवर केल्या जाणाऱ्या ऑपरेशन्स निर्दिष्ट करतात.

पूर्वीच्या आणि आत्ताच्या इलेक्ट्रॉनिक घटकांची तुलना केल्यास, साईझ, विजेचा वापर आणि कार्यक्षमता या सर्वांमध्ये सुधारणा होत आहेत आणि याचे एक प्राथमिक कारण म्हणजे संपूर्ण प्रणाली लहान आहे आणि त्याची रचना एकाच चिपमध्ये ठेवतात. 1970 मध्ये, मायक्रोचिप तयार करण्यात आली आणि प्रोसेसरचे सर्व घटक सिलिकॉनच्या एकाच तुकड्यावर ठेवण्यात आले. अशा प्रकारे, चिपचा साईझ कमी होतो आणि त्याचा वेग वाढतो. या मायक्रोचिपच्या निर्मितीसह, मायक्रोप्रोसेसरमध्ये मायक्रो हा वर्ड वापरून मायक्रोप्रोसेसर चा जन्म झाला.

मायक्रोप्रोसेसर हे एक प्रोग्राम करण्यायोग्य उपकरण आहे जे संख्या घेते, मेमोरी तील प्रोग्रामनुसार अरीथमेटिक आणि लॉजिक कार्ये करतात आणि परिणाम निर्माण करतात. उदाहरणार्थ, समजा एक सर्किट 1s इनपुट बिट स्ट्रीम स्कॅन करण्यासाठी तयार केले आहे आणि नमुना सलग चार 1 बिट त्यानंतर 0 शोधण्यासाठी तयार केले आहे. आउटपुट बिट 0 वर सेट केल्याशिवाय ते 1 आउटपुट करेल. फ्लिप फ्लॉप आणि गेट वापरून तयार केलेले सर्किट हे विशिष्ट कार्य करू शकते. सर्किट प्रोग्राम करण्यायोग्य नाही कारण ते केवळ एकाच अनुप्रयोगाशी संबंधित विशिष्ट ऑपरेशन्स करू शकतात. याउलट, मायक्रोप्रोसेसर ही प्रोग्रामेबल उपकरणे आहेत. दिलेल्या प्रोग्रॅममध्ये दिलेल्या इंस्ट्रक्शनच्या क्रमाच्या आधारे, ते त्यांना प्राप्त झालेल्या माहितीवर कामकाजाच्या विविध स्थिती अंमलात आणू शकतात. तर, मायक्रोप्रोसेसर इंस्ट्रक्शन द्वारे चालवले जातात. व्यावहारिकदृष्ट्या प्रत्येक मायक्रोप्रोसेसरमध्ये उपकरणाला त्याच्या प्रारंभिक स्थितीत परत आणण्यासाठी रीसेट पिन असते. हे विशिष्ट ठिकाणी मेमोरीमध्ये शोधणे सुरू करेल आणि तेथून रीसेट इंस्ट्रक्शन पुनर्प्राप्त करेल. ती इंस्ट्रक्शन अंमलात आणते, नंतर त्याचे प्रोग्राम काउंटर आणि रजिस्टर प्रकार अद्ययावत करते जेणेकरून ते पुढील इंस्ट्रक्शनचा संदर्भ देईल, इत्यादी.

आधुनिक मायक्रोप्रोसेसर खोलीचे तापमान नियमन स्कॅन करू शकतात. हे प्रथम तापमान संवेदक वाचन वाचते, नंतर सुरक्षित तापमान व्हॅल्यू आणि त्यावरील विचलनाच्या आधारे वातानुकूलित यंत्र किंवा हीटर्स चालू करतात. जर त्याच मायक्रोप्रोसेसरचा वापर तापमान देखरेखीशिवाय इतर कशासाठीही केला गेला असेल, जसे की कन्व्हेयर बेल्ट मॉनिटरिंग. हे अनुप्रयोग कन्व्हेयर बेल्ट कंट्रोलद्वारे चालवले जातात. अशा परिस्थितीत, मायक्रोप्रोसेसर अपरिवर्तित राहील; फक्त ते चालवत असलेले सॉफ्टवेअर बदलेल. तर आता मेमोरीमध्ये एक वेगळा प्रोग्राम रन



स्कॅन करा
मायक्रोप्रोसेसरच्या
परिचयासाठी

केल्या जाईल आणि CPU काहीतरी वेगळे करेल. मायक्रोप्रोसेसरचा प्रोग्राम करण्यायोग्य घटक काहीतरी वेगळे करण्यासाठी प्रोग्राम विभागात बदल करू शकतो.

मायक्रोप्रोसेसरची रचना विशिष्ट इंस्ट्रक्शन किंवा कार्ये पार पाडण्यासाठी केली जाते. याला इंस्ट्रक्शन सेट असे म्हणतात. मायक्रोप्रोसेसर चे युजर मान्युअल स्पष्ट करते की हा CPU कोणत्या कमांड ला सपोर्ट देईल. अशा इंस्ट्रक्शन चा वापर केवळ प्रोग्राम तयार करण्यासाठी केला जातात. प्रोग्राम्स C, JAVA किंवा C++ सारख्या (high level language) उच्च-स्तरीय भाषेत लिहिले जातात आणि नंतर संकलित केले जातात आणि अंतर्निहित प्रोसेसरला समजेल अशा भाषेत अनुवादित केले जातात.

8085 सीपीयू असलेल्या संगणकाच्या बाबतीत, C प्रोग्राम कार्यान्वित होईल. 8085- कॉम्पॅटिबल कोड संकलकाद्वारे तयार केला जाईल. जर तोच प्रोग्राम ARM प्रोसेसरवर चालवायचा असेल, तर कंपाइलर सोर्स कोडला इंस्ट्रक्शन सेट मध्ये रूपांतरित करतो जो ARM CPU द्वारे वाचला आणि समजला जाऊ शकतो.

मायक्रोप्रोसेसरचा डिझायनर मशीन इंस्ट्रक्शन सेट समजून घेण्यासाठी तो प्रोग्राम करू शकतो. नमूद केलेल्या मार्गदर्शक तत्वांनुसार वापरकर्त्यांनी त्यांचा स्वतःचा कोड तयार करणे आवश्यक आहे. सीपीयू मेमोरी आणि आय/ओ उपकरणांसारख्या इतर घटकांशी जोडला जाऊ शकतो.

प्रोग्रामसाठी इनपुट व्हॅल्यू मेमोरी मध्ये जतन केली गेल्यास मेमोरी इनपुट डिव्हाइस म्हणून वापरली जाऊ शकते. उदाहरणार्थ, जर 100 संख्यांची क्रमवारी (sorting) लावण्याचे काम असेल आणि 100 संख्या मेमोरी मध्ये साठवल्या गेल्या असतील. हे मेमोरी मध्ये जतन केलेल्या 100 संख्यांवर कार्य करेल. दुसरीकडे, जर तापमान संवेदक स्थापित आणि चालू असेल तर प्रणालीने त्या तापमान संवेदकाकडून तापमान घेतले पाहिजे. तर, इनपुट डिव्हाइसेस भिन्न असू शकतात आणि इनपुट डिव्हाइसमधून इनपुट उद्भवू शकते.

ही उपकरणे बाहेरील जगाची माहिती प्रणालीमध्ये आणतात. म्हणूनच, हा बाहेरील जगाशी असलेला संवाद आहे आणि अडचण अशी आहे की बाहेरील जग बहुतेक ॲनालॉग आहेत. जरी डिजिटल सर्किट्स डिजिटल डेटासह कार्य करणाऱ्या मायक्रोप्रोसेसरसाठी तयार केले गेले असले तरी बाह्य वातावरण ॲनालॉग आहेत. तर, विशिष्ट स्विच चालू किंवा बंद आहे की नाही किंवा विशिष्ट ब्लब चालू किंवा बंद आहे की नाही यासारख्या अत्यंत सोप्या प्रकारच्या इनपुट घेतल्याने सतत डेटा मिळतो. तर, उदाहरणार्थ, तापमान संवेदक त्याला प्राप्त होत असलेल्या तापमानाच्या आधारे व्होल्टेजची श्रेणी तयार करेल. त्यामुळे ते डिजिटल नाही.

इनपुटसाठी ॲनालॉग-टू-डिजिटल कन्व्हर्जन (ADC) आणि इनपुट उपकरणापासून प्रोसेसरपर्यंतच्या आउटपुटसाठी डिजिटल-टू-ॲनालॉग कन्व्हर्जन (DAC) सारखे अनेक डेटा कन्व्हर्टर असतात. काही प्रोसेसर या डेटा कन्व्हर्टरसह सुसज्ज आहेत. तर सीपीयू केवळ डिजिटल डेटावर प्रक्रिया करतो, परंतु या एडीसी आणि डीएसी कन्व्हर्टरद्वारे ॲनालॉग वातावरणाशी संवाद साधू शकतो. तर, ही उपकरणे बाहेरील जगातून डेटा प्रणालीमध्ये आणतात आणि कीबोर्ड, माऊस आणि स्विच यासारखी उपकरणे ही सर्व इनपुट उपकरणे मानली जातात.

संख्यांमध्ये कसे बदल केले जातात याचे वर्णन पुढील भागात केले आहे. त्याची संभाव्य व्हॅल्यू कोणती असू शकतात? त्यांची संख्या खूप मोठी असू शकते. उदाहरणार्थ, पूर्णांक संख्यांची व्हॅल्यू अनंतापर्यंत असू शकतात आणि ती अनियंत्रितपणे प्रचंड असू शकतात, परंतु जेव्हा ती संगणकात प्रविष्ट केली जाते, तेव्हा ती अमर्यादित मेमोरी

वापरून संग्रहित केली जाऊ शकत नाही. त्यामुळे आकाराची (size) मर्यादा असेल. 16-बिट साइन केलेल्या पूर्णांकाची श्रेणी-32768 ते + 32767 दरम्यान असू शकते, तर अनसाइन केलेल्या संख्येची संख्या 0 ते 65,535 दरम्यान असू शकते. म्हणून, एक मर्यादित श्रेणी आहे.

अशा प्रकारे, मायक्रोप्रोसेसरला आयुष्याबद्दल फारच मर्यादित दृष्टीकोन आहे, कारण तो केवळ बायनरी संख्या, म्हणजे, 0 आणि 1 ला समजून घेतो आणि ऑक्टल किंवा हेक्साडेसिमल पूर्णांक समजून घेत नाही. जरी बायनरी संख्या अनेकदा आपल्या स्वतःच्या आकलनार्थ ऑक्टल आणि हेक्साडेसिमल संकेतांचा वापर करून चित्रित करतात, तरी बायनरी संख्या प्रणाली नेहमीच सीपीयू च्या आत वापरली जाते. आणि बायनरी अंक, ज्याला बिट म्हणून देखील ओळखले जाते, ते संख्या सेटयित करण्यासाठी वापरले जाते कारण बिट 1 उच्च व्होल्टेज व्हॅल्यू म्हणून किंवा लॉजिक लेव्हल उच्च म्हणून संग्रहित केले जाते, तर बिट 0 लॉजिक लेव्हल लो दर्शविते.

मायक्रोप्रोसेसर बिट्सचे ग्रुप ओळखतो; बिट्सचे ग्रुप हे वर्ड म्हणून ओळखले जातात. एक वर्ड (word) 16 किंवा 32 सारख्या विशिष्ट संख्येच्या बिटांनी बनलेला असू शकतो. याउलट, जेव्हा ते बेरीज, वजाबाकी, गुणाकार, भागाकार आणि तुलना यासारख्या प्राथमिक क्रिया करते, तेव्हा ते वर्ड स्तरावर कार्य करते. प्रोसेसरचा डिझायनर त्याद्वारे प्रोसेसरच्या वर्ड चा साईझ ठरवीतो. अशाप्रकारे, मायक्रोप्रोसेसरच्या बाबतीतही हेच लागू होतात.

मायक्रोप्रोसेसर वर्डमधील बिट्सची संख्या त्याची क्षमता दर्शवते. मायक्रोप्रोसेसरमधील 8-बिट सीपीयू 8-बिट व्हॅल्यूवर प्रक्रिया करू शकतो. पूर्णांक व्यतिरिक्त, केवळ 8-बिट व्हॅल्यू वापरली जातात, ज्याचा परिणाम एकूण 9 बिटसाठी 8 बिट अधिक 1 बिट कॅरी असतो. जर बेसिक प्रोसेसर 8-बिट एडिशनला सपोर्ट करत असेल आणि त्याला 16-बिट एडिशन करणे आवश्यक असेल. 8-बिट जोड वापरून सॉफ्टवेअरमध्ये 16-बिट जोड तयार करणे वर्ड आहे. तथापि, प्रोसेसर 16-बिट वर्ड हाताळतो आणि एकाच ऑपरेशनमध्ये 16-बिट जोडणी केली जाते त्यापेक्षा जास्त वेळ लागेल. आधुनिक मायक्रोप्रोसेसरमध्ये 32-बिट आणि 64-बिट वर्ड साईझ असतात. त्यामुळे, ते बरेच मोठे डेटा सेट व्यवस्थापित करू शकते.

8085, 8088, मोटोरोला 6800 आणि 6800 हे 8-बिट वर्ड ओळखणारे सर्वात जुने मायक्रोप्रोसेसर आहेत. हे मायक्रोप्रोसेसर केवळ 8-बिट शब्दांना आधार देणारे आहेत. अशा प्रकारे, 8086 आणि 68000 सारखे प्रोसेसर 16-बिट वर्ड सह विकसित केले गेले आहेत. आता, हे लक्षात घेणे महत्वाचे आहे की इंटेल 8086 आणि 8088 अस्तित्वात आहेत. तर, 8088 हे 8086 नंतर का आले, जसे नाव सांगते, आणि 8085 हा प्रोसेसर 8 बिट्स का आहे तर 8086 हे 16 बिट्स का आहे. याचे कारण असे आहे की 8085 हे 8086 आणि 8088 च्या आधी आले होते, ते अत्यंत लोकप्रिय होते आणि अनेक प्रणाली तयार करण्यासाठी वापरले जात होते. यात 8-बिट वर्ड आहेत, जे प्रोसेसरच्या डेटा लाईन्सवरील पिनच्या संख्येशी संबंधित आहेत. अशा प्रकारे, हे 8 बिट्स होतात.

जेव्हा 8086 बाजारात आणले गेले तेव्हा त्याने 8085 चीपची जागा 8086 चीपने घेतली कारण डेटाचा साईझ स्वतःच 8 बिट्सवरून 16 बिट्समध्ये बदलला आहे.



स्कॅन करा

8085 मायक्रोप्रोसेसर
बद्दल अधिक जाणून
घेण्यासाठी

त्यामुळे यंत्रसामग्री विसंगत झाली आहे. जेव्हा हे लक्षात आले, तेव्हा इंटेलने एक पाऊल मागे घेतले आणि 8-बिट सीपीयू 8088 तयार केले, ज्याचे अंतर्गत कार्य 8086 सारखेच आहे. बाहेरून, तथापि, त्यात 8-बिट वर्ड इंटरफेस आहे, म्हणून सर्व 8085-आधारित प्रणालींना या 8088 प्रोसेसरने बदलले जाऊ शकते, तर हार्डवेअर समान राहते. सॉफ्टवेअर अद्ययावत करणे आवश्यक होते कारण ते आता 8088 शी सुसंगत होते, परंतु हार्डवेअरमध्ये बदल करण्याची आवश्यकता नव्हती.

8086 मायक्रोप्रोसेसरमध्ये 16 बिट असलेले वर्ड आहेत. आधुनिक प्रोसेसर 32-बिट आणि 64-बिट वर्ड आहेत. या 16-बिट वर्ड शी संबंधित कोणतेही प्रमाणीकरण किंवा मानक अस्तित्व नाही. पारंपारिकपणे, 8 बिट्स एक बाइट बनवतात, परंतु वर्ड ची लांबी प्रोसेसरवर अवलंबून असते. जर एखादा वर्ड 16 बिट्स लांब असेल तर 8 बिट्सचा गट हॉल्फ-वर्ड किंवा बाइट म्हणून ओळखला जातो आणि 4 बिट्सचा गट निब्वल म्हणून ओळखला जातो. तर, 16-बिट वर्ड साठी, चार निब्वल्स आहेत. तर, 32-बिट गटांना कधीकधी मोठे वर्ड म्हणून संबोधले जाते. वर्ड चा साईझ प्रमाणित केला जात नाही कारण तो प्रोसेसरच्या वर्ड च्या आकारानुसार निर्धारित केला जातो.



स्कॅन करा
इंस्ट्रक्शन सेट आर्किटेक्चर
उत्क्रांतीसाठी

इतर मायक्रोप्रोसेसर आहेत जे एकाच वेळी 64, 80 किंवा 128 बिट्स हाताळू शकतात. सध्या, सर्व प्रोसेसर एका वेळी किमान 32 बिट्स हाताळतात. अशा प्रकारे, खूप वाढीव क्षमतेसह उत्कृष्ट प्रोसेसर तयार करणे शक्य आहे, ज्यामुळे एकाच इंस्ट्रक्शनसह मोठे डेटा बिट्स व्यवस्थापित करता येतात.

3.2 इंस्ट्रक्शन सेट आर्किटेक्चर

संगणकाच्या डारेक्टिव संचाला एकतर "कॉम्प्लेक्स इंस्ट्रक्शन सेट कॉम्प्युटर (CISC)" किंवा "रिड्यूस्ड इंस्ट्रक्शन सेट कॉम्प्युटर (RISC)" म्हणून वर्गीकृत केले जाऊ शकते. मशीन लँगवगे प्रोग्रॅम हे प्रोसेसरच्या इंस्ट्रक्शन संचानुसार तयार केले जातात. सुरुवातीच्या संगणकांमध्ये इंस्ट्रक्शन चे लहान आणि सोपे सेट होतात कारण ते चालवण्यासाठी त्यांना शक्य तितके कमी हार्डवेअर वापरावे लागत होते. जेव्हा इंटिग्रेटेड सर्किट्स येतात आणि डिजिटल हार्डवेअर स्वस्त होतात, तेव्हा संगणकाच्या इंस्ट्रक्शन सामान्यतः अधिकाधिक गुंतागुंती चे होतात. कधीकधी अनेक संगणकांच्या डारेक्टिव संचांमध्ये 200 पेक्षा जास्त इंस्ट्रक्शन असतात.

हे संगणक डेटा स्वरूपांची विस्तृत श्रेणी आणि संबोधित करण्याची यंत्रणा देखील वापरतात. CISC आधारित संगणकावर मोठ्या संख्येने इंस्ट्रक्शन असतात. 1980 च्या दशकाच्या सुरुवातीला, RISC इंस्ट्रक्शन सेट संरचनेची रचना कमी इंस्ट्रक्शन सह करण्यात आली होती. आरआयएससी (RISC) ची साधी आणि सरळ रचना, त्याच्या मर्यादित मेमोरी प्रवेशामुळे, त्याला प्रोसेसरमध्ये वेगाने कार्यान्वित करण्यास अनुमती देतात.

3.2.1 CISC ची वैशिष्ट्ये

CISC संरचनेचे प्रमुख पैलू खालीलप्रमाणे आहेत:

1. इंस्ट्रक्शन ची मोठी संख्या, बहुतेकदा 100 ते 250 दरम्यान असते.

2. काही कमांड जे विशिष्ट कार्ये पूर्ण करतात आणि बऱ्याचदा क्वचितच वापरल्या जातात.
3. अडुरेसींग मोड ची एक विस्तृत श्रेणी -सामान्यतः 5 ते 20 भिन्न मोड मध्ये.
4. बदलत्या लांबीसह इंस्ट्रक्शन चे स्वरूप.
5. इंस्ट्रक्शन मेमोरी तील ऑपरेंड मानिप्युलेट करू शकतात.

संगणकाच्या इंस्ट्रक्शन सेट ची रचना मशीन भाषेचे घटक आणि उच्च-स्तरीय प्रोग्रामिंग भाषांच्या वापरावरील निर्बंध या दोन्हींचा विचार करतात. उच्च-स्तरीय प्रोग्रामिंगचे मशीन लॅंग्वेज कोडमध्ये भाषांतर करण्यासाठी कम्पालर चा वापर केला जातो. गुंतागुंतीच्या इंस्ट्रक्शन सेट मुळे कम्पिलेशन सोपे होते आणि संगणकाची एकूण गती वाढते. कारण स्टेटमेंट स्वयंचलितपणे मशीन इंस्ट्रक्शन नुसार केले जातात. आयबीएम 370 संगणक हे CISC आर्किटेक्चरचे एक उदाहरण आहे.

CISC बदलत्या लांबीच्या इंस्ट्रक्शन स्वरूपांचा वापर करते. इंस्ट्रक्शन ऑपरेंड केवळ दोन बाइट लांब असतात, तर दोन मेमोरी ऍड्रेसचा संदर्भ देणाऱ्या इंस्ट्रक्शन पाच बाइटपर्यंत लांब असू शकतात.

एका फिक्स्ड लांबीच्या मेमोरी वर्डमध्ये विविध प्रकारच्या इंस्ट्रक्शन बसवण्यासाठी वर्डच्या आतील बाइट मोजण्यासाठी एक विशेष डीकोडिंग सर्किट आवश्यक आहे. CISC च्या इंस्ट्रक्शन मेमोरीमध्ये साठवलेल्या ऑपरेंडमध्ये थेट बदल करण्यास परवानगी देतात. अतिरिक्त इंस्ट्रक्शन आणि अडुरेसींग मोड अंमलात आणण्यासाठी अधिक हार्डवेअर सर्किटरीची आवश्यकता आहे, ज्यामुळे प्रणालीची कार्यक्षमता कमी होऊ शकते.

3.2.2 RISC ची वैशिष्ट्ये

खालील RISC स्थापत्यशास्त्रीय वैशिष्ट्ये संगणकाचा डारेक्टिव सेट कमी करून अंमलबजावणीचा वेळ कमी करतात:

1. इंस्ट्रक्शन आणि अडुरेसींग मोड कमी आहेत.
2. केवळ "LOAD" आणि "STORE" इंस्ट्रक्शनद्वारे मेमोरीमध्ये प्रवेश केला गेला जातो.
3. सर्व कार्ये सीपीयू रजिस्टर वापरून केली जातात. प्रक्रिया युनिटमध्ये मोठ्या संख्येने रजिस्टर आहेत.
4. एका फिक्स्ड लांबीसह इंस्ट्रक्शन स्वरूप जे सहजपणे डीकोड केले जाऊ शकते. त्यामुळे RISC पाइपलाइनची रचना कार्यक्षमतेने केली जाऊ शकते.
5. इंस्ट्रक्शनची अंमलबजावणी एकाच क्लॉक सायकल मध्ये होते.
6. कंट्रोल युनिट हे मायक्रोप्रोग्राम करण्याऐवजी हार्डवायर्ड असते.

RISC प्रोसेसरकडे केवळ मेमोरी प्रवेशासाठी "लोड" आणि "स्टोअर" इंस्ट्रक्शन असतात आणि ते मुख्यतः रजिस्टर-ते-रजिस्टर कार्यांवर अवलंबून असतात. "लोड" इंस्ट्रक्शनद्वारे ऑपरेंड योग्य CPU रजिस्टरमध्ये लोड केले जाते. परिणाम मेमोरीमध्ये हलविण्यासाठी "स्टोअर" इंस्ट्रक्शन वापरली जाते. अंतरिम परिणाम संचयीत करण्यासाठी आणि इतर रजिस्टरमध्ये डेटा ट्रान्सफरस गती



स्कॅन करा

RISC विरुद्ध CISC च्या तुलनात्मक अभ्यासासाठी

देण्यासाठी, मोठ्या संख्येने रजिस्टर आवश्यक आहेत. सर्वात वारंवार विनंती केलेल्या ऑपरेंडस रजिस्टरमध्ये ठेवून, रजिस्टर-टू-मेमोरी व्यवहार कमी केले जाऊ शकतात.

प्रत्येक क्लॉक सायकल मध्ये, RISC प्रोसेसर इंस्ट्रक्शनच्या पाईपलाईन अंमलबजावणीद्वारे एक इंस्ट्रक्शन कार्यान्वित करू शकतो. मेमोरी ऍक्सेसला रजिस्टर ऑपरेशनसपेक्षा जास्त वेळ लागत असल्याने, 'लोड' आणि 'स्टोअर' इंस्ट्रक्शनना मेमोरीशी संवाद साधण्यासाठी दोन क्लॉक सायकल ची आवश्यकता असते.

RISC रजिस्टर हाताळणी सुलभ करून डारेक्टिव सेट रचना सुलभ करते. कारण व्यावहारिकदृष्ट्या सर्व इंस्ट्रक्शन मूलभूत रजिस्टर ऍड्रेसचा वापर करतात. तात्काळ ऑपरेंड आणि रिलेटिव्ह मोड यासारख्या काही अडुसेसिंग मोड पुरविल्या जाऊ शकतात. वाजवी मूलभूत इंस्ट्रक्शन संरचनेचा वापर करून इंस्ट्रक्शन ची लांबी फिक्स्ड केली जाऊ शकते. CISC इंस्ट्रक्शनचे स्वरूप डीकोड करणे सोपे आहे. इंस्ट्रक्शन आणि त्यांचे स्वरूप सुव्यवस्थित करून कंट्रोल लॉजिक सोपे केले जाऊ शकते. जलद ऑपरेशनसाठी हार्डवायर्ड कंट्रोल युनिटला प्राधान्य दिले जातात.

3.3 अभ्यागत प्रोग्रॅमची रचना तत्त्वे

उच्च-स्तरीय प्रोग्राम्सचे मशीन-रीडेबल इंस्ट्रक्शनमध्ये भाषांतर करताना प्रोग्रामर्सना अधिक स्वातंत्र्य आणि निवडी देण्यासाठी इंस्ट्रक्शन स्वरूप आणि अडुसेसिंग पद्धतीमध्ये विविध वैशिष्ट्ये प्रदान केली जातात.

3.3.1 इंस्ट्रक्शन फॉरमाट:

प्रत्येक इंस्ट्रक्शन कोडचा अर्थ खाली सूचीबद्ध केल्याप्रमाणे CPU मधील विशिष्ट स्वरूपाद्वारे लावला जातो.

1. ऑपरेशन कोडसाठी एक फील्ड, जे ऑपरेशनचा प्रकार दर्शवते.
2. मेमोरी स्थान किंवा प्रोसेसर रजिस्टरचा ऍड्रेस निर्दिष्ट करण्यासाठी वापरले जाणारे फील्ड.
3. मोड फील्ड ऑपरेंड किंवा प्रभावी ऍड्रेस संगणनाचा दृष्टीकोन निर्धारित करते.

काही प्रकरणांमध्ये, शिफ्ट-प्रकारच्या इंस्ट्रक्शनमध्ये शिफ्टची संख्या निर्दिष्ट करणे यासारखी आणखी विशेष फील्ड वापरली जाऊ शकतात. ऑपरेशन कोड फील्डमध्ये जोडणे, वजा करणे, पूरक करणे आणि शिफ्ट करणे यासारख्या अनेक प्रोसेसर ऑपरेशनसचे वर्णन केले जाते. मोड फील्ड निर्दिष्ट ऍड्रेसवरून ऑपरेंड निवडण्यासाठी अनेक पर्याय दर्शविते.

संगणकांमध्ये विविध पत्त्यांसह बदलत्या लांबीच्या इंस्ट्रक्शन असू शकतात. संगणकाच्या इंस्ट्रक्शन स्वरूपातील ऍड्रेस फील्डची संख्या रजिस्टरच्या अंतर्गत संरचनेवर अवलंबून असते. बहुतेक संगणक तीन सीपीयू संस्थांच्या प्रकारांपैकी एकाचे असतात:

1. **सिंगल अक्युमुलेटर ऑर्गनाइझेशन:** प्रत्येक कृती इम्प्लिसिट अक्युमुलेटर रजिस्टर वापरून अंमलात आणली जाते (A). या प्रकारच्या संगणकाच्या इंस्ट्रक्शन स्वरूपामध्ये एकच ऍड्रेस फील्ड असते. सामान्य उदाहरणांपैकी

एक म्हणजे ADD X इंस्ट्रक्शन, जी $A \rightarrow A + M[X]$ म्हणून बेरीज करते. जेथे $M[X]$ हा मेमोरीमध्ये एक्स स्थानावर संग्रहित केलेला डेटा आहेत.

2. **सामान्य रजिस्टर ऑर्गनाइझेशन:** तीन रजिस्टर आणि दोन ऍड्रेस फील्ड ही सामान्य रजिस्टर प्रकारच्या संरचनेची उदाहरणे आहेत. उदाहरणार्थ, तीन रजिस्टर फील्ड ADD R4, R5, R6 म्हणून दर्शविले जातात. हे सूचित करते की ADD ऑपरेशन रजिस्टर R5 आणि R6 दरम्यान केले जाते आणि परिणाम रजिस्टर R4, i.e., $R4 \leftarrow R5 + R6$ मध्ये संग्रहित केला जातो.

डेस्टिनेशन रजिस्टर आणि सोर्स रजिस्टर समान असल्यास, इंस्ट्रक्शन दोन रजिस्टर फील्डद्वारे दर्शविली जाऊ शकते. उदाहरणार्थ, ADD R4, R5 ही इंस्ट्रक्शन ऑपरेशन $R4 \leftarrow R4 + R5$ दर्शवते. या इंस्ट्रक्शनमध्ये, फक्त R4 आणि R5 साठी रजिस्टर ठिकाणे पुरवली गेली पाहिजेत.

ट्रान्सफर इंस्ट्रक्शन, अर्थात, MOV इंस्ट्रक्शन देखील दोन ऍड्रेस फील्ड वापरतात. उदाहरणार्थ, संगणकावर अवलंबून, MOV R4, R5 हे $R4 \leftarrow R5$ (रजिस्टर R5 रजिस्टर R4 वर डेटा ट्रान्सफर करते) किंवा $R5 \leftarrow R4$ (रजिस्टर R4 रजिस्टर R5 वर डेटा ट्रान्सफर करते) रजिस्टर दरम्यान डेटा ट्रान्सफर करतात. सामान्य उद्देशासाठी संगणक दोन किंवा तीन ऍड्रेस फील्ड इंस्ट्रक्शन वापरू शकतात.

3. **स्टॅक ऑर्गनायझेशन:** स्टॅक-ऑर्गनाइज्ड CPU मध्ये PUSH आणि POP ऍड्रेस-फील्ड इंस्ट्रक्शन असतात. PUSH X स्टॅकच्या वर वर्ड घालतो आणि स्टॅक पॉइंटर स्वयंचलितपणे अद्ययावत होतो. इतर कार्ये थेट डारेक्टिवद्वारे केली जाऊ शकतात आणि या इंस्ट्रक्शनसाठी ऍड्रेस फील्ड निर्दिष्ट करणे आवश्यक नसते. उदाहरणार्थ, ADD इंस्ट्रक्शन स्टॅकमधून वरच्या दोन व्हॅल्यूना POP करते, त्यांना जोडते आणि नंतर बेरीज स्टॅकवर स्टोर करतात.

3.3.2 ऍड्रेसिंग मोड

इंस्ट्रक्शनचे ऑपरेंड फील्ड रजिस्टर किंवा मेमोरीमध्ये संचयीत केलेल्या डेटावर केले जाणारे ऑपरेशन परिभाषित करते [1]. प्रोग्रॅमच्या अंमलबजावणी दरम्यान ऑपरेंडचे व्हॅल्यू डारेक्टिवच्या ऍड्रेसिंग मोडद्वारे निर्धारित केले जाते. ऑपरेंडमध्ये प्रवेश करण्यापूर्वी डारेक्टिवच्या ऍड्रेस क्षेत्राचा अर्थ कसा लावायचा हे ऍड्रेसिंग मोड निर्दिष्ट करते.

संगणक खालीलप्रमाणे विविध अड्रेसिंग पद्धती वापरू शकतात:

1. विविध वैशिष्ट्यांची उपलब्धता कोड लिहिण्यात लवचिकता प्रदान करते जसे की डेटा अनुक्रमणिका, लूप नियंत्रणासाठी काउंटर, मेमोरी पॉइंटर आणि प्रोग्रामचे रेलोकेशन.
2. असेंब्ली लॅंग्वेज प्रोग्रामरसाठी उपलब्ध असलेल्या अनेक अड्रेसिंग पद्धतींचा लाभ घेऊन डारेक्टिवची संख्या आणि अंमलबजावणीचा वेळ कमी करणे शक्य आहे. अशा प्रकारे, डारेक्टिवच्या ऍड्रेस क्षेत्रात आवश्यक बिट्सची संख्या कमी केल्याने प्रोग्रॅमची कार्यक्षमता सुधारते.

प्रोसेसरचे कंट्रोल युनिट इंस्ट्रक्शन सायकल म्हणून ओळखले जाणारे ऑपरेशन करते, ज्यामध्ये मेमोरीमधून इंस्ट्रक्शन वाचणे, इंस्ट्रक्शन डीकोड करणे आणि डीकोड केलेला प्रोग्राम कार्यान्वित करणे समाविष्ट आहे.

प्रोग्राम काउंटर (PC) हा एक विशेष संगणक रजिस्टर आहे जी मेमोरीतील प्रोग्राममधील प्रत्येक इंस्ट्रक्शन किती वेळा अंमलात आणली गेली आहे हे मोजते. जेव्हा एखादी इंस्ट्रक्शन मेमोरीतून पुनर्प्राप्त केली जाते, तेव्हा त्याचा ऍड्रेस संगणकात साठवला जातो आणि संगणकात एकाने वाढ केली जाते. डिकोडिंगच्या दुसऱ्या टप्प्यात, डारेक्टिवचे कार्य, अडुरेसींग मोड आणि ऑपरेंडची ठिकाणे फिक्स्ड केली जातात. इंस्ट्रक्शनवर प्रक्रिया केल्यानंतर, संगणक अनुक्रमाने पुढील इंस्ट्रक्शन शोधण्यासाठी स्टेप 1 वर परत जातो. काही संगणकांमध्ये डारेक्टिवची अडुरेसींग पद्धती निर्दिष्ट करण्यासाठी स्वतंत्र बायनरी कोड वापरणे ही सामान्य प्रथा आहे, जसे ऑपरेशन कोड निर्दिष्ट करण्यासाठी करतात. इतर प्रणालींमध्ये, डारेक्टिवची पद्धत आणि प्रकार दोन्ही एकाच बायनरी कोडद्वारे नियुक्त केले जातात. इंस्ट्रक्शन निर्दिष्ट करण्यासाठी अनेक भिन्न अडुरेसींग मोड वापरल्या जाऊ शकतात आणि अनेकदा अनेक अडुरेसींग मोड एकाच इंस्ट्रक्शनमध्ये मिसळल्या जातात.

इंस्ट्रक्शनमध्ये ऍड्रेस फील्ड समाविष्ट असू शकते किंवा नसूही शकते. उपस्थित असलेली कोणतीही ऍड्रेस फील्ड एकतर विशिष्ट मेमोरी स्थान किंवा विशिष्ट CPU रजिस्टरकडे निर्देशित केली जाऊ शकतात. ऑपरेंडच्या मेमोरी ऍड्रेसमध्ये मुळात डिसप्लेसमेंट, बेस ऍड्रेस आणि इंडेक्स व्हॅल्यू [2] असते. याव्यतिरिक्त, इंस्ट्रक्शनमध्ये अनेक ऍड्रेसची फील्ड असू शकतात, ज्यापैकी प्रत्येक भिन्न भिन्न ऍड्रेस पद्धत वापरू शकते.

1. इम्प्लाइड मोड: कोणतेही ऑपरेंड स्पष्टपणे निर्दिष्ट केलेले नाही म्हणून हे इम्प्लाइड मोड ऍड्रेसिंग आहे.

उदाहरणार्थ,

INC

इंस्ट्रक्शन **INC** अक्युमुलेटर तील कन्टेन्ट वाढवते. स्टॅक-आधारित संगणक सामान्यतः हे ऍड्रेसिंग मोड वापरतात.

2. इमेडिएट मोड: ऑपरेंड निर्देशात स्पष्टपणे दिले आहे. रजिस्टर्स स्थिर व्हॅल्यूवर सेट करण्यासाठी तात्काळ-मोड इंस्ट्रक्शन वापरल्या जाऊ शकतात.

उदाहरणार्थ,

MOV R4, #300

MOV इंस्ट्रक्शन मध्ये, एक ऑपरेंड हे व्हॅल्यू आहे म्हणून हे इमेडिएट मोड आहे. **MOV** इंस्ट्रक्शन 300 ची व्हॅल्यू रजिस्टर R4 मध्ये ठेवते. व्हॅल्यूच्या समोरील संख्या चिन्ह (#) सूचित करते की हे व्हॅल्यू इमेडिएट ऑपरेंड म्हणून वापरले जाणार आहे.

3. रजिस्टर मोड: रजिस्टर मध्ये ऑपरेंड असतात. रजिस्टर हे एकतर इंस्ट्रक्शन चे पहिले किंवा दुसरे ऑपरेंड असू शकते. उपलब्ध रजिस्टर्सच्या संचातून विशिष्ट रजिस्टरची निवड केली जाते.

उदाहरणार्थ,

MOV R1, R4

MOV डारेक्टिवमधील दोन्ही ऑपरेंड हे रजिस्टर आहेत म्हणून हे रजिस्टर ऍड्रेसिंग मोड आहे. **MOV** इंस्ट्रक्शन रजिस्टर R4 ची सामग्री रजिस्टर R1 मध्ये साठवते.

4. एक्सोल्यूट (डायरेक्ट) मोड: डारेक्टिवच्या ऍड्रेस क्षेत्रात ऑपरेंडचा प्रभावी (effective) ऍड्रेस असतो.

ऑपरेंड मेमोरीमध्ये साठवला जातो. ऑपरेंडमध्ये प्रवेश करण्यासाठी इंस्ट्रक्शन विशिष्ट ऍड्रेस प्रदान करतात.

उदाहरणार्थ,

MOV R1, LOC

MOV इंस्ट्रक्शन मेमोरी लोकेशन LOC पासून व्हॅल्यू रजिस्टर R2 मध्ये ठेवते.

5. **रजिस्टर इनडायरेक्ट मोड:** मेमोरी ऍड्रेस किंवा रजिस्टर सामग्री ऑपरेंडचा प्रभावी ऍड्रेस निर्दिष्ट करते. उदाहरणार्थ, `MOV R1, (R4)`

रजिस्टर R4 चा मजकूर हा मेमोरी ऍड्रेस आहे जेथे ऑपरेंड व्हॅल्यू संग्रहित केले जाते. रजिस्टर इनडायरेक्ट मोड मेमोरीमध्ये ऍड्रेस देखील प्रदान करू शकतो जिथे ऑपरेंड व्हॅल्यूचा ऍड्रेस संग्रहित केला जातो.

उदाहरणार्थ, `MOV R1, (X)`

मेमोरीमधील ऑपरेंड व्हॅल्यू Y ऍड्रेसवर संग्रहित केले जाते आणि हा Y ऍड्रेस मेमोरीमध्ये X स्थानावर संग्रहित केला जातो. येथे, रजिस्टरऐवजी, मेमोरी स्थान X मध्ये ऑपरेंडचा ऍड्रेस आहे.

6. **इंडेक्स ऍड्रेसिंग मोड:** या ऍड्रेसिंग मोडची सिंटॅक्स $X(R_i)$ किंवा $X[R_i]$ आहे, ऍड्रेस $X + R_i$, येथे X ऑफसेट आहे. जर वाक्यरचना (R_i) किंवा $[R_i]$ असेल तर R_i ला संबोधित करा जेथे ऑफसेट (विस्थापन) = 0 आहे. इंडेक्स रजिस्टर हा एक विशेष सीपीयू रजिस्टर आहे जी इंडेक्स व्हॅल्यू ठेवण्यासाठी वापरली जाते.

उदाहरणार्थ, `MOV AL, [DI + 2]`

`MOV AL, [DI + 2]` इंस्ट्रक्शन `MOV AL, 2 DI` म्हणून देखील दर्शविली जाऊ शकते. मेमोरीमध्ये संचयीत केलेले ऑपरेंड व्हॅल्यू मिळविण्यासाठी, [] मधील रजिस्टरमध्ये विस्थापन (ऑफसेट) क्रमांक 2 जोडला जातो. हे देखील शक्य आहे की विस्थापन रजिस्टरमधून वजा केले जाऊ शकते, उदाहरणार्थ,

`MOV AL, [SI-1].`

उदाहरण 3.1

दिलेल्या इंस्ट्रक्शनच्या संबोधित करण्याच्या पद्धती फिक्सड करा.

(1) ADD (2) COM (3) ADD R1, R2 (4) MOV R1, #100 (5) LOAD R2, NUM

उत्तर:

(1) इम्प्लाईड ऍड्रेसिंग मोड.

स्टॅक-आधारित संगणकांमध्ये, शून्य ऍड्रेस इंस्ट्रक्शन वापरल्या जातात. परिणामी, एडीडी इंस्ट्रक्शन स्टॅकमधून वरच्या दोन इंस्ट्रक्शन बाहेर काढतात (पॉप), जोडा आणि नंतर परिणाम परत स्टॅकच्या शीर्षस्थानी ढकलतात.

(2) इम्प्लाईड ऍड्रेसिंग मोड.

सेट्यकातील मजकुराची पूर्तता करा

(3) रजिस्टर ऍड्रेसिंग मोड.

कारण डारेक्टिवमधील नॉट ऑपरेंड हे रजिस्टर आहेत. इंस्ट्रक्शन ADD R1 आणि R2 रजिस्टरच्या सामग्रीची भर घालते आणि परिणाम R1 रजिस्टरमध्ये संग्रहित केला जातो.

(4) इमेडिएट ऍड्रेसिंग मोड.

व्हॅल्यू 100 ची उपस्थिती सूचित करते की हे व्हॅल्यू इमेडिएट ऑपरेंड म्हणून वापरले जाणार आहे. MOV इंस्ट्रक्शन 100 ची किंमत रजिस्टरकृत R1 मध्ये ठेवते.

(5) एक्सोल्यूट अॅड्रेसिंग मोड.

LOAD ही इंस्ट्रक्शन मेमोरी लोकेशन NUM पासून व्हॅल्यू रजिस्टर R2 मध्ये लोड करते.

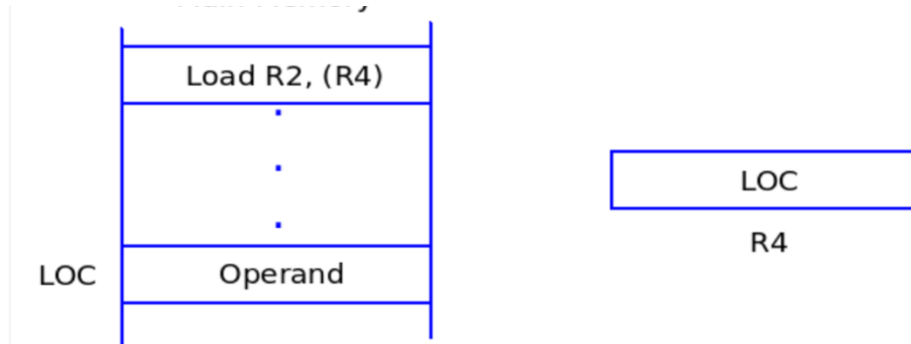
उदाहरण 3.2

दिलेल्या इंस्ट्रक्शनच्या संबोधित करण्याच्या पद्धती फिक्स्ड करा.

(1) LOAD R2, (R4) (2) LOAD R2, (A)

उत्तर: (1) रजिस्टर इन्डायरेक्ट अॅड्रेसिंग मोड:

ऑपरेंडचा ऍड्रेस रजिस्टरकृत R4, मध्ये संग्रहित आहे. i.e. LOC



(2) रजिस्टर इन्डायरेक्ट अॅड्रेसिंग मोड:

ऑपरेंडचा ऍड्रेस मेमोरी ऍड्रेस A, मध्ये संग्रहित केला जातो. i.e LOC

	.
	LOAD R2,(A)
	.
A	LOC
	.
LOC	Operand
	.

आकृती. 3.1: ऑपरेंड ऍड्रेस रजिस्टरमध्ये साठवला जातो आणि ऍड्रेस मेमोरीमध्ये साठवला जातो तेव्हा अप्रत्यक्ष अॅड्रेसिंग मोडची रजिस्टर करा.

7. बेस-इंडेक्स अॅड्रेसिंग मोड:

बेस-इंडेक्स अॅड्रेसिंगची सिंटॅक्स (Ri, Rj) किंवा [Ri, Rj], प्रमाणे आहे. ऍड्रेस Ri + Rj बेस रजिस्टर आणि डिस्प्लेसमेंट रजिस्टरची बेरीज हा ऑपरेंडचा प्रभावी ऍड्रेस आहे. बेस-इंडेक्स अॅड्रेसिंग मोड वेगवेगळ्या मेमोरी विभागांमधील प्रोग्रामचे ऍड्रेसतर सुलभ करतो.

उदाहरणार्थ,

MOV AL, (BL, DI)

MOV AL, (BL, DI) डारेक्टिवचे स्पष्टीकरण MOV AL, [BL, DI] मेमोरीमध्ये ऑपरेंडचा ऍड्रेस मोजण्यासाठी विस्थापन रजिस्टर DI, i.e., [BL + DI] मध्ये बेस रजिस्टर BL जोडला जातो.

8. बेस-इंडेक्स ऑफसेट:

या अॅड्रेसिंग मोडचा सिंटॅक्स X (Ri,Rj) किंवा X[Ri,Rj], सारखेच आहे. ऍड्रेस X+Ri+Rj आहे. उदाहरणार्थ, MOV AL, X(BL, DI) येथे X हे ऑफसेट (स्थिर) व्हॅल्यू आहे. मेमोरीमध्ये ऑपरेंडचा ऍड्रेस मिळविण्यासाठी हा X [BL + DI + X] मध्ये जोडला जातो. ही लवचिकता रजिस्टरमधील अनेक घटकांमध्ये प्रवेश करण्यासाठी उपयुक्त आहे.

उदाहरण 3.3

दिलेल्या इंस्ट्रक्शनच्या संबोधित करण्याच्या पद्धती फिक्सड करा.

(1) LOAD R2,50 (R4) जेव्हा R4 = 2000

(2) LOAD R2, (R4, R5)

(3) LOAD R2,10 (R4, R5)

उत्तर:

(1) इंडेक्स अॅड्रेसिंग मोड:

	LOAD R2,50(R4)
	.
2000	LOC
	.
2050	Operand

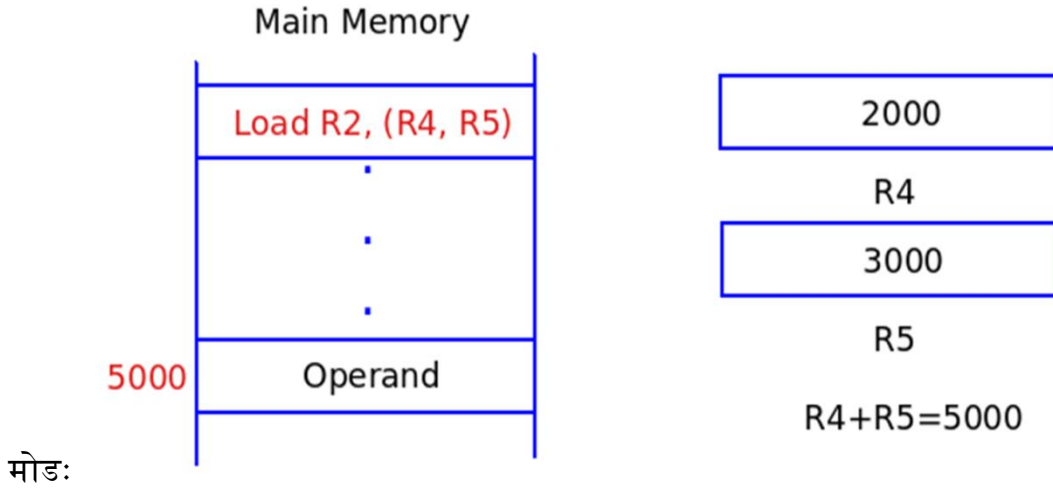
OFFSET= 50

2000

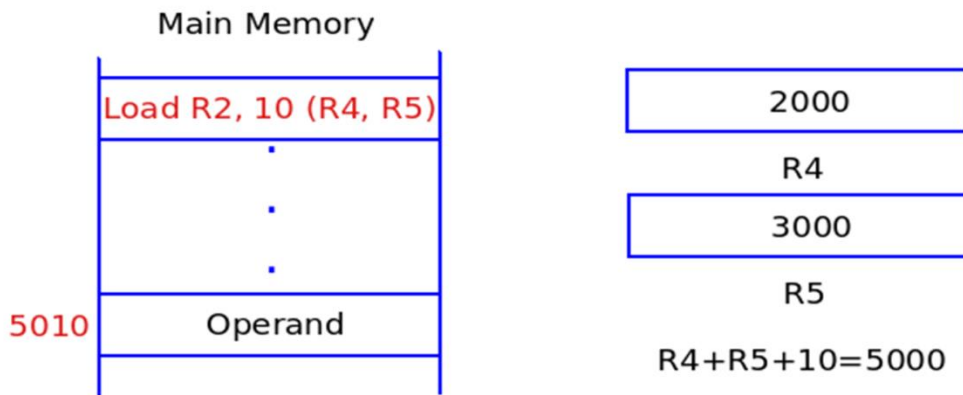
R4

आकृती. 3.2: मुख्य मेमोरी आणि अनुक्रमणिका पत्तन पद्धतीमध्ये मजकूर नोंदवा

(2) बेस-इंडेक्स अड्रेसिंग



(3) बेस-इंडेक्स ऑफसेट अड्रेसिंग मोड:



आकृती. 3.3 बेस-इंडेक्स आणि बेस-इंडेक्स ऑफसेट अड्रेसिंग मोडमध्ये मुख्य मेमोरी आणि रजिस्टर सामग्री

9. रिलेटिव्ह अड्रेसिंग मोड: प्रोग्राम काउंटरचा वापर करून रिलेटिव्ह अड्रेसिंगची गणना केली जाते. जेव्हा अनुक्रमणिका व्हॅल्यू प्रोग्राम काउंटरवर जोडले जाते, तेव्हा प्रभावी ऍड्रेस पुढील डारेक्टिवच्या ऍड्रेसशी संबंधित असतो. या ऍड्रेसच्या पद्धतीचा वाक्यरचना $X(PC)$ किंवा $X[PC]$, ऍड्रेस $PC + X$ आहे.

उदाहरणार्थ,

$$\text{ब्रँच} > 0 \text{ LOC}$$

हा ऍड्रेस प्रोग्राम काउंटरवरून ऑफसेट म्हणून मोजला जातो.

10. ऑटो-इंक्रिमेंट मोड: याला ऑटो पोस्ट-इंक्रिमेंट असेही म्हणतात. हे रजिस्टर इनडायरेक्ट अड्रेसिंग मोड वापरते आणि त्यानंतरच्या इन्स्ट्रक्शनमध्ये मेमोरीमधील पुढील वर्डमध्ये प्रवेश करण्यासाठी रजिस्टर वाढवते. ऑटो-इंक्रिमेंट अड्रेसिंग मोडसाठी वाक्यरचना $(Ri)+$, ऍड्रेस $Ri + 1 \times L$ आहे, जेथे L ही बाइटमधील मेमोरी वर्डची लांबी आहे.

उदाहरणार्थ,

$$ST (R1) +, R5$$



स्कॅन करा
8086 अड्रेसिंग
पद्धतींसाठी

ही इंस्ट्रक्शन रजिस्टर R1 मध्ये असलेल्या मेमोरी ऍड्रेसमध्ये रजिस्टर R5 ची सामग्री जतन करते, नंतर स्वयंचलित वाढ अंमलबजावणीमुळे पुढील मेमोरी स्थान दर्शविणे सुरू करते. अशा प्रकारे पुढील मेमोरी स्थानाचा ऍड्रेस आता रजिस्टर R1 मध्ये संग्रहित केला जातो. हे ऍड्रेसिंग मोड सलग मेमोरी ऍड्रेसवरून डेटा आयटममध्ये प्रवेश करण्यासाठी उपयुक्त आहे.

11. ऑटो-डिक्रीमेंट मोड: याला ऑटो प्री-डिक्रीमेंट असेही म्हणतात. ऑटोडिक्रीमेंट ऍड्रेसिंग मोडचा वाक्यरचना- (Ri) ऍड्रेस $Ri-1 \times L$ आहे, येथे L ही बाइटमधील मेमोरी वर्डची लांबी आहे. हे रजिस्टर इनडायरेक्ट ऍड्रेसिंग मोड वापरते आणि नंतर सध्याच्या इंस्ट्रक्शनमध्येच मेमोरीमधील मागील वर्डमध्ये प्रवेश करण्यासाठी रजिस्टर कमी करते. उदाहरणार्थ,

ST-(R1), R5

ही इंस्ट्रक्शन प्रथम रजिस्टर R1 मध्ये साठवलेले स्थान कमी करते. आता रजिस्टर R1 कडे मागील मेमोरी स्थान आहे जेथे रजिस्टर R5 ची सामग्री संग्रहित केली आहे. हे ऍड्रेसिंग मोड मागील मेमोरी ऍड्रेसवरून डेटा आयटममध्ये प्रवेश करण्यासाठी उपयुक्त आहे.

उदाहरण 3.4

दिलेल्या इंस्ट्रक्शनच्या संबोधित करण्याच्या पद्धती फिक्स्ड करा. समजा मेमोरीतील प्रत्येक वर्डची लांबी 4 बाइट आहे.

MOV R2, (R4) +

ADD R3, R4

ADD R6, -(R5)

R2, R3 आणि R6 या रजिस्टरमध्ये काय समाविष्ट आहे?

उत्तर:

MOV इंस्ट्रक्शनमध्ये ऑटो-इंक्रीमेंट ऍड्रेसिंग मोड आहे. पहिल्या ADD इंस्ट्रक्शनमध्ये रजिस्टर इनडायरेक्ट ऍड्रेसिंग मोड आहे आणि दुसऱ्या ADD इंस्ट्रक्शनमध्ये ऑटो-डिक्रीमेंट ऍड्रेसिंग मोड आहे.

Main Memory

	MOV R2, (R4) +
	ADD R3, R4
	ADD R6, -(R5)
100	10
104	20
108	30

100

R4

MOV R2, (R4) +

104

ADD R3, R4

R4

ADD R6, -(R5)

108

R5

104

R5

आकृती. 3.4: रजिस्टर एंड्रेसमध्ये ऑटोइन्क्रीमेंट आणि ऑटोडेक्रीमेंट अँड्रेसिंग मोडसह सुधारणा

तिन्ही इंस्ट्रक्शन अंमलात आणल्यानंतर, आकृती 3.4 मध्ये दर्शविल्याप्रमाणे R2 = 10, R3 = 20 आणि R6 = 20 ही रजिस्टर व्हॅल्यू आहेत.

3.4 8086 मायक्रोप्रोसेसर आर्किटेक्चर:

8086 हा 16-बिट मायक्रोप्रोसेसर आहे. मायक्रोप्रोसेसर 20-बिट अँड्रेस बसचा वापर करून एकाच वेळी मेमोरीमध्ये 16 बिट डेटा वाचू किंवा लिहू शकतो. या मायक्रोप्रोसेसरद्वारे जास्तीत जास्त 1 एमबी रॅम वापरली जाऊ शकते.

3.4.1.8086 मायक्रोप्रोसेसर कार्यात्मक एकक:

एक्झिक्युशन युनिट (EU) आणि बस इंटरफेस युनिट (BIU) हे आकृती 3.5 मध्ये दर्शविल्याप्रमाणे 8086 मायक्रोप्रोसेसर आर्किटेक्चरचे प्रमुख कार्यात्मक युनिट आहेत.



स्कॅन करा

8086 मायक्रोप्रोसेसरचे
कार्यात्मक भाग

- **एक्झिक्युशन युनिट (EU)**

एक्झिक्युशन युनिट बीआययू (BIU) ला माहिती कुठे मिळवायची याची इंस्ट्रक्शन देते आणि त्यानंतर त्या इंस्ट्रक्शनचे डीकोडिंग आणि अंमलबजावणी करण्यासाठी बीआययू जबाबदार असते. एएलयू आणि इंस्ट्रक्शन डीकोडर वापरून डेटा ऑपरेशन्स व्यवस्थापित करणे हे त्याचे काम आहे. प्रणाली बस थेट अंमलबजावणी युनिटशी जोडलेल्या नाहीत.

❖ **फ्लॅग रजिस्टर:** हा 16-बिटचा रजिस्टर आहे. त्यात दोन प्रकारचे फ्लॅग असतात- कंडिशनल फ्लॅग आणि कंट्रोल फ्लॅग. फ्लॅगची स्थिती रजिस्टरमध्ये साठवलेल्या व्हॅल्यूच्या आधारे बदलते.

➤ **कंडिशनल किंवा स्टेटस फ्लॅग**

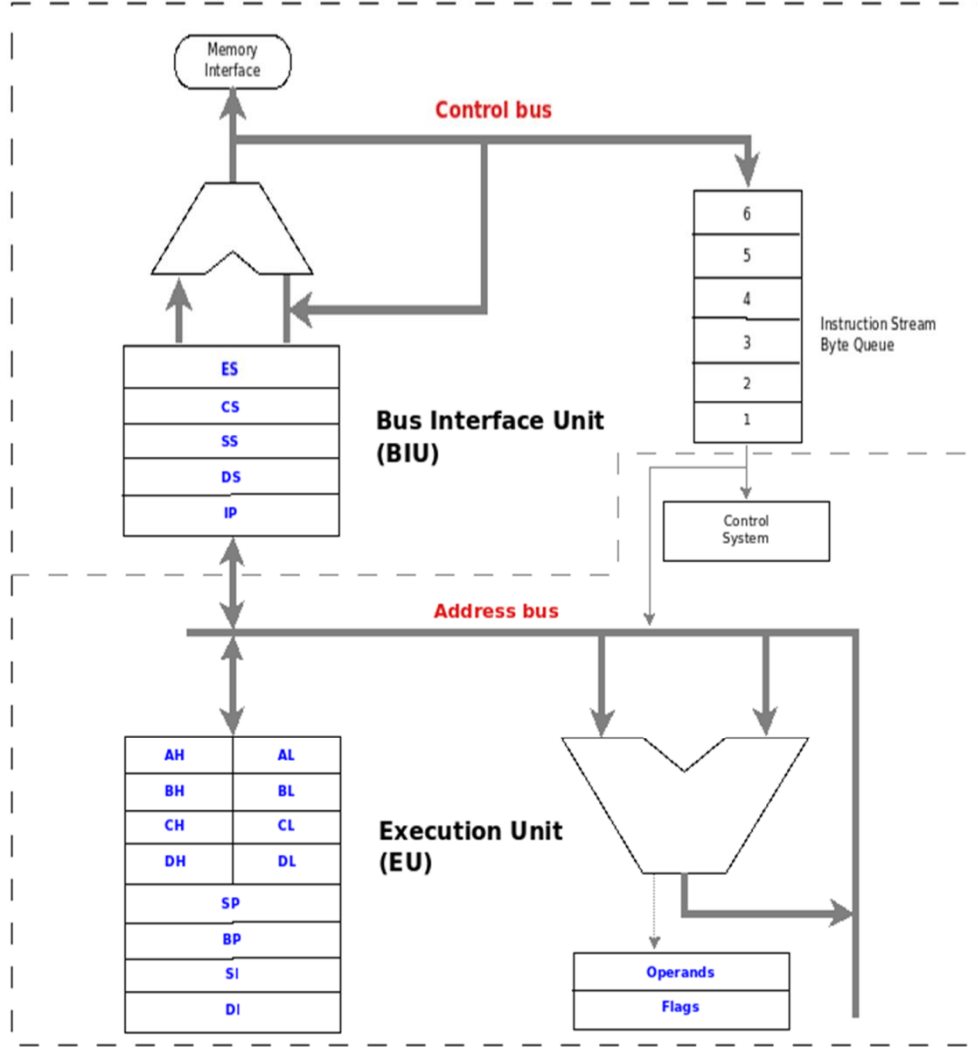
खालील कंडिशनल फ्लॅग पूर्वी अंमलात आणलेल्या एरिथमेटिक किंवा लॉजिक कार्याच्या परिणामाचे प्रतिनिधित्व करतात:

1. **कॅरी फ्लॅग (C):** दोन n-बिट बायनरी संख्या जोडताना, निकाल n-बिटपेक्षा जास्त असल्यास, कॅरी फ्लॅग $C = 1$ म्हणून सेट होतो, अन्यथा, $C = 0$.
2. **ऑक्सिलरी फ्लॅग (AF):** BCD क्रमांकांसाठी, ऑक्सिलरी फ्लॅग CF सारखाच असतो. दोन संख्या आणि कॅरी जोडताना खालच्या निम्नलपासून उच्च निम्नलपर्यंत तयार होते, तर $AF = 1$. अन्यथा, $AF = 0$.
3. **ओव्हरफ्लो फ्लॅग (O):** C फ्लॅग आणि ओव्हरफ्लो फ्लॅग समान आहेत. साइन केलेल्या पूर्णांकावरील कोणत्याही एरिथमेटिक किंवा लॉजिक ऑपरेशनचा परिणाम रजिस्टरच्या निर्दिष्ट क्षमतेपेक्षा जास्त असल्यास, तो 1 वर सेट केला जातो. नसल्यास, O फ्लॅग 0 वर सेट केला आहे.
4. **पॅरिटी फ्लॅग (P):** पॅरिटी फ्लॅग सम आहे. जर बायनरी व्हॅल्यूतील 1 ची संख्या सम असेल तर $P = 1$ सेट करा, अन्यथा $P = 0$.
5. **झिरो फ्लॅग (Z):** जर सेटयक रजिस्टरमधील एरिथमेटिक किंवा लॉजिक ऑपरेशनचा परिणाम शून्य असेल तर $Z = 1$ सेट करा, अन्यथा $Z = 0$.
6. **साईन फ्लॅग (S):** एरिथमेटिक किंवा लॉजिक क्रियांनंतर, परिणामाचे चिन्ह सकारात्मक असल्यास, $S = 1$ सेट करा, अन्यथा $S = 0$.

➤ **कंट्रोल फ्लॅग:** इंटरप्ट फ्लॅग, डायरेक्शनल फ्लॅग आणि ट्रॅप फ्लॅग हे कंट्रोल ध्वज आहेत.

1. **इंटरप्ट फ्लॅग (I):** जेव्हा बाह्य उपकरणे मायक्रोप्रोसेसरला इंटरप्ट कंट्रोल सिग्नल पाठवतात, तेव्हा $I = 1$; अन्यथा, $I = 0$.
2. **डायरेक्शनल फ्लॅग (D):** जेव्हा स्ट्रिंगला मेमोरीमध्ये वरच्या बाइटपासून खालच्या बाइटपर्यंत प्रवेश केला जातो तेव्हा डायरेक्शनल फ्लॅग 1 वर सेट केला जातो. जेव्हा स्ट्रिंगला मेमोरीमध्ये खालच्या बाइटवरून उच्च बाइटमध्ये प्रवेश केला जातो तेव्हा ते 0 वर सेट केले जाते.

3. ट्रॅप फ्लॅग (T): ऑन-चिप डीबगिंग ट्रॅप फ्लॅग वापरून केले जाते. जेव्हा $T = 1$ असते, तेव्हा कार्य सिंगल-स्टेप मोडमध्ये चालते. प्रत्येक इंस्ट्रक्शन अंमलात आणल्यावर अंतर्गत इंटरप्ट निर्माण होतो. यामुळे इंस्ट्रक्शनची अनुक्रमिक अंमलबजावणी शक्य होते.



आकृती. 3.5: 8086 मायक्रोप्रोसेसर आर्किटेक्चर

❖ जनरल परपज रजिस्टर:

8086 मायक्रोप्रोसेसरमध्ये 'AX', 'BX', 'CX' आणि 'DX' असे चार 16-बिट सामान्य-उद्देश रजिस्टर आहेत आणि आठ 8-बिट सामान्य-उद्देश रजिस्टर आहेत; ते आहेत ('AH', 'AL') ('BH', 'BL') ('CH', 'CL') आणि ('DH', 'DL') या रजिस्टरचा वापर एरिथमेटिक किंवा लॉजिक कार्यांदरम्यान डेटा संचयीत करण्यासाठी केला जातो.

1. अक्युमुलेटर रजिस्टर (AX)

'AX' रजिस्टरची क्षमता 16 बिट्सची आहे. ते दोन 8-बिट रजिस्टरमध्ये विभागले गेले आहे. हे अनुक्रमे "AH". आणि "AL". रजिस्टर आहेत. हे एरिथमेटिक किंवा लॉजिक क्रियांपूर्वी डेटा संचयीत करते. निकाल 'AX' रजिस्टरमध्ये उपलब्ध आहे.

2. बेस रजिस्टर (BX)

बेस रजिस्टर हे 16-बिटचे रजिस्टर आहे. याचा वापर व्हॅल्यूचा ऑफसेट ऍड्रेस सेटयित करण्यासाठी केला जातो. (operand). यात दोन 8-बिट रजिस्टर आहेत. (BH and BL). खालील उदाहरणात, 300H हा व्हॅल्यूचा ऑफसेट ऍड्रेस आहे आणि तो BL रजिस्टरमध्ये संचयीत केला जातो.

```
MOV bl, [300] (bl ← 300H)
```

3. काउन्ट रजिस्टर (CX)

काउन्टची रजिस्टर हा 16-बिट रजिस्टर असते. हे CH आणि CL नावाच्या दोन 8-बिट रजिस्टरमध्ये विभागले गेले आहे. याचा वापर लूप इंस्ट्रक्शनसाठी केला जातो. खालील उदाहरणात, रजिस्टर cx ची व्हॅल्यू 0 पर्यंत पोहोचेपर्यंत लूपची पुनरावृत्ती केली जाते.

```
MOV CX, 08
```

```
LOOP
```

4. डेटा रजिस्टर (DX)

डेटा रजिस्टर हे 16-बिटचे रजिस्टर आहे. हे DH आणि DL या दोन 8-बिट रजिस्टर मध्ये देखील विभागले गेले आहे. हा रजिस्टर विभागणी आणि गुणाकार कार्यासाठी वापरला जातो. उर्वरित व्हॅल्यू DX रजिस्टरमध्ये साठवली जाते आणि भाग AX रजिस्टरमध्ये साठवली जाते.

5. स्टॅक पॉइंटर (SP)

हा एक 16-बिट रजिस्टर आहे जो स्टॅक सेगमेंटचा सर्वात वरचा स्टॅक घटक दर्शविण्यासाठी वापरला जातो.

6. बेस पॉइंटर (BP)

बेस पॉइंटर रजिस्टर देखील स्टॅक सेगमेंटसाठी एक पॉइंटर आहे. हे 16-बिट रजिस्टर स्टॅकद्वारे पास केलेल्या घटकांमध्ये प्रवेश करण्यासाठी वापरले जाते.

7. सोर्स इंडेक्स (SI)

सोर्स इंडेक्स रजिस्टर हे 16-बिटचे रजिस्टर आहे. हे डेटाचे पॉइंटर अॅड्रेसिंग वापरते. ते डेटा विभागाकडे डारेक्टिव करते.

8. डेस्टिनेशन इंडेक्स (DI)

डेस्टिनेशन इंडेक्स रजिस्टर हा 16-बिटची रजिस्टर आहे. डेटाच्या पॉइंटर अॅड्रेसिंगमध्ये "DI" रजिस्टरचा वापर केला जातो.

9. अरीथमेटिक अँड लॉजिक युनिट (ALU)

सर्व एरिथमेटिक आणि लॉजिक क्रिया ALU परिपथामध्ये केल्या जातात.

• बस इंटरफेस युनिट (BIU)

बस इंटरफेस युनिटमध्ये सेगमेंट रजिस्टर, इन्स्ट्रक्शन क्यू, कंट्रोल युनिट आणि ऍड्रेस कॅल्कुलेशन साठी प्रक्रिया युनिट आहे. BIU हे सिस्टम बसद्वारे मेमोरी इंटरफेसशी जोडलेले आहे.

❖ इन्स्ट्रक्शन क्यू

बस इंटरफेस युनिटमध्ये 6-बाइट इन्स्ट्रक्शन क्यू आहे. बस इंटरफेस युनिट डारेक्टिवच्या रांगेत मेमोरीच्या पुढील इन्स्ट्रक्शन पुनर्प्राप्त करते आणि संग्रहित करते. एक्सिक्युशन युनिट डारेक्टिवच्या रांगेत वेगाने कार्यान्वित होते.

❖ सेगमेंट रजिस्टर

बस इंटरफेस युनिटमध्ये चार सेगमेंट रजिस्टर आहेत-"CS", "DS", "SS" आणि "EX" जे डेटा आणि इन्स्ट्रक्शनसाठी मेमोरी अॅड्रेसिंग सेटयित करतात जे प्रोसेसर मेमोरी पत्त्यांमध्ये प्रवेश करण्यासाठी वापरतो.

❖ कोड सेगमेंट (CS)

एक्झिक्युटेबल प्रोग्राम 16-बिट कोड सेगमेंट रजिस्टरमध्ये संग्रहित केला जातो. या रजिस्टरमध्ये मेमोरीच्या कोड विभागाचा ऍड्रेस असतो.

❖ डेटा सेगमेंट (DS)

डेटा सेगमेंट रजिस्टर 16-बिट अॅड्रेसिंग ची 64 KB प्रोग्राम मेमोरी असते. डेटा विभागात सामान्य रजिस्टर ("AX", "BX", "CX", "DX") आणि डारेक्टिवक रजिस्टर ("SI", "DI") संदर्भ डेटा. POP आणि LDS इन्स्ट्रक्शन डेटा विभागात थेट बदल करू शकतात.

❖ अतिरिक्त सेगमेंट (ES)

अतिरिक्त सेगमेंट हा 64KB सेगमेंट ऍड्रेससह 16-बिट रजिस्टर आहे. स्ट्रिंग ES मध्ये डेस्टिनेशन स्थानाची अतिरिक्त माहिती संग्रहित करते.

❖ स्टॅक सेगमेंट (SS)

कार्यान्वित करताना, डेटा आणि अॅड्रेसिंग संचयीत करण्यासाठी 16-बिट स्टॅक सेगमेंट रजिस्टर वापरून मेमोरी व्यवस्थापित केली जाते.

❖ इन्स्ट्रक्शन पॉइंटर (IP)

16-बिट इन्स्ट्रक्शन पॉइंटर (IP) रजिस्टर मध्ये पुढील इन्स्ट्रक्शनचा ऍड्रेस असतो.

- **इफेक्टिव्ह ऍड्रेस कॅल्क्युलेशन**

मायक्रोप्रोसेसर 16-बिट ऍड्रेससह डेटामध्ये प्रवेश करू शकत नाही कारण मेमोरीची साठवण क्षमता 1 MB आहे. प्रत्येक मेमोरीमध्ये 20-बिट ऍड्रेस असतो, त्यामुळे मेमोरीतून डेटा किंवा इन्स्ट्रक्शनमध्ये प्रवेश करण्यापूर्वी, इफेक्टिव्ह किंवा फिजिकल ऍड्रेस मोजला जातो.

फिजिकल ऍड्रेस = सेगमेंट रजिस्टर x 10H + ऑफसेट

येथे सेगमेंट रजिस्टर हे CS आहे आणि ऑफसेट हा इन्स्ट्रक्शन ऑपरेंड ऍड्रेस आहे.

- **कंट्रोल युनिट (CU)**

कंट्रोल युनिट सीपीयू रजिस्टर आणि ALU दरम्यान डेटा ट्रान्सफरचे समन्वय करते. कंट्रोल युनिट संगणकाच्या प्रत्येक भागावर कंट्रोल ठेवते, सर्व भाग, ट्राफिक इत्यादींचे समन्वय साधते.

3.4.2 8086 मधील इन्स्ट्रक्शन चे प्रकार:

8086 डारेक्टिव संचाद्वारे समर्थित आठ प्रकारच्या डेटा ट्रान्सफर इन्स्ट्रक्शन, लॉजिक इन्स्ट्रक्शन, एरिथमेटिक इन्स्ट्रक्शन, स्ट्रिंग मानिपुलेशन इन्स्ट्रक्शन, प्रोसेस कंट्रोल इन्स्ट्रक्शन आणि कंट्रोल ट्रान्सफर इन्स्ट्रक्शनचा समावेश आहे. त्यापैकी काही आपल्याला परिचित आहेत, जसे की डेटा ट्रान्सफर, एरिथमेटिक आणि लॉजिक. याउलट, स्ट्रिंग मानिपुलेशन इन्स्ट्रक्शन अगदी नवीन आहेत. याव्यतिरिक्त, विशिष्ट कंट्रोल ट्रान्सफर इन्स्ट्रक्शनमध्ये ब्रांच कॉल इत्यादींचे विविध प्रकार समाविष्ट असतील, ज्यामुळे ते प्रोसेसर-विशिष्ट बनतील.

• **डेटा ट्रान्सफर इन्स्ट्रक्शन** सोर्सकडून डेस्टिनेशनस्थानावर डेटा ट्रान्सफर करतात.

1. MOV सोर्सकडून डेस्टिनेशन स्थानावर माहिती ट्रान्सफर करते. उदाहरणार्थ, MOV ax, bx हा bx रजिस्टरमधून ax रजिस्टरमध्ये डेटा ट्रान्सफर करतो.
2. PUSH वर्ड स्टॅकच्या शीर्षस्थानी ठेवतो.
3. POP स्टॅकच्या वरच्या भागातून वर्ड काढून टाकते.
4. XCHG दोन रजिस्टर्समध्ये डेटाची देवाणघेवाण करते.

• **इनपुट/आउटपुट पोर्ट ट्रान्सफर इन्स्ट्रक्शन** खालीलप्रमाणे आहेत.

1. IN प्रदान केलेल्या पोर्टपासून रजिस्टरपर्यंत वर्ड/बाइट रीड करतात.
2. OUT सेटयक रजिस्टरतून प्रदान केलेल्या पोर्टवर वर्ड/बाइट लिहितो.
3. LEA संचालन ऍड्रेस प्रदान केलेल्या रजिस्टरवर भारित करतात.
4. LDS मेमोरीमधून डेटा सेगमेंट रजिस्टर आणि इतर रजिस्टर लोड करतात.
5. LES मेमोरीमधून अतिरिक्त सेगमेंट रजिस्टर आणि इतर रजिस्टर लोड करतात.

● **एरिथमेटिक इंस्ट्रक्शन** अशा प्रकारे परिभाषित केल्या जाऊ शकतात

❖ **बेरीज**

1. ADD- हे दुसऱ्या बाइट/वर्डमध्ये बाइट/वर्ड जोडण्यासाठी वापरले जाते.
2. ADC- कॅरीसह बेरीज करा.
3. INC- बाइट/वर्ड 1 ने वाढवला. हे फक्त एडिशन ऑपरेशन आहे.

❖ **वजा करणे**

1. SUB- हे दुसऱ्या वर्ड/बाइट मधून वर्ड/बाइट वजा करण्यासाठी वापरले जाते.
2. SBB- बारो सह वजाबाकी.
3. DEC- 1 ने वजा केलेला वर्ड किंवा बाइट.
4. CMP- दोन रेजिस्टर्स ऑपरेंड वजाबाकी केलेले आणि शून्याच्या तुलनेत निकाल.

❖ **गुणाकार**

1. MUL- दोन अन-साईन्ड संख्यांचा गुणाकार करण्या करीता.
2. IMUL- दोन साईन्ड संख्यांच्या गुणाकार करण्या करीता.

❖ **भागाकार**

1. DIV- दोन अन-साईन्ड संख्यांच्या भागाकार करण्या करीता.
2. IDIV- दोन साईन्ड केलेल्या संख्यांच्या भागाकार करण्या करीता.

● **बिट मानिपुलेशन इंस्ट्रक्शन**

या NOT, OR, AND, XOR इत्यादी लॉजिकल ऑपरेशन्स आहेत.

● **शिफ्ट इंस्ट्रक्शन**

डावीकडे शिफ्ट इंस्ट्रक्शन - प्रत्येक बिट डावीकडे आणि सर्वात उजव्या बिट स्थितीत शून्य जोडला जातो. उदाहरणार्थ shl al, 1 जेथे, AL चे सर्व बिट्स एका स्थानावरून डावीकडे हलवले जातात आणि सर्वात उजव्या स्थानावर शून्य जोडले जाते.

शिफ्ट राईट इंस्ट्रक्शन - सर्व बिट्स उजवीकडे हलवले जातात आणि सर्वात डाव्या बिट पोजिशनमध्ये शून्य जोडले जाते. उदाहरणार्थ shr al, 2 जेथे, AL चे सर्व बिट्स उजवीकडे हलवले जातात आणि सर्वात डाव्या स्थितीत शून्य जोडले जाते.

अंकगणिती उजवी शिफ्ट - डावीकडील बिट स्थितीत समान राहते आणि सर्व बिट्स उजव्या एका स्थानाकडे हलविले जातात. उदाहरणार्थ, ashr al, 1 जेथे, AL चे बिट्स उजवीकडे हलवले जातात आणि सर्वात डावीकडे बिट व्हॅल्यू शेवटच्या बिट स्थितीत जोडली जाते.

● **ब्रांच इंस्ट्रक्शन** 8086 मध्ये खालील ब्रांच इंस्ट्रक्शन वापरल्या गेल्या आहेत:

1. JA साइन न केलेल्या संख्यांची तुलना करते. CMP किंवा SUB डारेक्टिवनंतर, जर पहिला ऑपरेंड दुसऱ्यापेक्षा मोठा किंवा समान असेल तर JA इंस्ट्रक्शन ताबडतोब लेबलवर कंट्रोल ट्रान्स्फर करते.

2. समान असल्यास JE लेबलवर उडी मारते.
 3. कॅरी फ्लॅग = 1 असल्यास जेसी उडी मारते.
 4. जर शून्य फ्लॅग = 1 असेल तर JZ उडी मारतो.
- लूप इंस्ट्रक्शन cx 0 पर्यंत पोहोचेपर्यंत लूप इंस्ट्रक्शन पुनरावृत्ती केली जाते.

युनिट सारांश:

- मायक्रोप्रोसेसर ही एएलयू (एरिथमेटिक लॉजिक युनिट), रजिस्टर आणि कंट्रोल सर्किटरी असलेली सिलिकॉन चिप आहे. सेंट्रल प्रोसेसिंग युनिट (सीपीयू) मध्ये एरिथमेटिक लॉजिक कार्ये करण्यासाठी एरिथमेटिक लॉजिक युनिट, मध्यवर्ती व्हॅल्यू संचयीत करण्यासाठी रजिस्टर बँक, ज्यातून सीपीयू अनेकदा ऑपरेंड ओढते आणि संपूर्ण प्रोसीजरचे नियमन करणारे कंट्रोल लॉजिक असते.
- 8085 आणि 8086 मायक्रोप्रोसेसर हे मानक 8-बिट आणि 16-बिट मायक्रोप्रोसेसर आर्किटेक्चर आहेत.
- मायक्रोप्रोसेसर हे एक प्रोग्रामेबल डिव्हाइस आहे जे संख्या घेते, मेमोरीमध्ये प्रोग्रामनुसार एरिथमेटिक आणि लॉजिक कार्ये करते आणि परिणाम तयार करते.
- संगणक दोन प्रकारचे डारेक्टिव सेट वापरतात: कॉम्प्लेक्स इंस्ट्रक्शन सेट कॉम्प्युटर (CISC) आणि रिड्यूस्ड इंस्ट्रक्शन सेट कॉम्प्युटर (RISC).
- CISC च्या इंस्ट्रक्शन बदलत्या लांबीच्या आणि गुंतागुंतीच्या इंस्ट्रक्शन स्वरूपाच्या असतात. या प्रकारच्या इंस्ट्रक्शन मेमोरीमध्ये साठवलेल्या ऑपरेंडमध्ये थेट बदल करण्यास अनुमती देतात.
- RISC प्रोसेसरकडे मेमोरी प्रवेशासाठी "load" आणि "store" इंस्ट्रक्शन असतात आणि ते मुख्यतः रजिस्टर-ते-रजिस्टर कार्यांवर अवलंबून असतात. load इंस्ट्रक्शन योग्य सीपीयू रजिस्टरमध्ये कार्य करते. "store" इंस्ट्रक्शन मेमोरीमध्ये परिणाम जतन करण्यासाठी वापरली जाते. बहुतेक संगणक तीन सीपीयू ऑर्गनाईझेशन च्या प्रकारांपैकी एकाशी संबंधित असतात: एकल संचायक ऑर्गनाईझेशन, सामान्य रजिस्टर ऑर्गनाईझेशन आणि स्टॅक ऑर्गनाईझेशन.
- ऑपरेंडमध्ये प्रवेश करण्यापूर्वी डारेक्टिवच्या ऍड्रेस क्षेत्राचा अर्थ कसा लावायचा हे ऍड्रेसिंग मोड निर्दिष्ट करते.
- डारेक्टिवचे ऑपरेंड फील्ड रजिस्टर किंवा मेमोरीमध्ये साठवलेल्या डेटावर केले जाणारे ऑपरेशन परिभाषित करते. 8086 हा 16-बिट मायक्रोप्रोसेसर आहे. मायक्रोप्रोसेसर 20-बिट ऍड्रेस बसचा वापर करून एकाच वेळी मेमोरीमध्ये 16 बिट डेटा वाचू किंवा लिहू शकतो. 8086 मायक्रोप्रोसेसर आर्किटेक्चरचे प्रमुख कार्यात्मक घटक म्हणजे एक्झिक्युशन युनिट (EU) आणि बस इंटरफेस युनिट (BIU).
- एक्झिक्युशन युनिट हे अनेक फ्लॅग रजिस्टर आणि विशेष उद्देश कंट्रोल फ्लॅगनी बनलेले असते. तर बस इंटरफेस युनिटमध्ये सेगमेंट रजिस्टर, एक इंस्ट्रक्शन रांग, एक कंट्रोल युनिट आणि ऍड्रेस गणना प्रक्रिया युनिट असते.

- एक्झिक्युशन युनिट BIU ला माहिती कुठे मिळवायची याची इंस्ट्रक्शन देते आणि त्यानंतर त्या इंस्ट्रक्शनचे डीकोडिंग आणि अंमलबजावणी करण्यासाठी BIU जबाबदार असते. 8086 च्या डारेक्टिव संचामध्ये डेटा ट्रान्सफर इंस्ट्रक्शन, लॉजिक इंस्ट्रक्शन, एरिथमेटिक इंस्ट्रक्शन, स्ट्रिंग मानिपुलेशन इंस्ट्रक्शन, प्रक्रिया कंट्रोल इंस्ट्रक्शन आणि कंट्रोल ट्रान्सफर इंस्ट्रक्शन यांचा समावेश आहे.

स्वाध्याय:

बहुपर्यायी प्रश्न

प्रश्न 3.1 बायनरी डेटाचे सर्वात लहान एकक.

- (अ) बिट (ब) बाइट (क) निब्ल (ड) वर्ड

प्रश्न 3.2 64 बिट वर्डमध्ये

- (अ) 4 बिट (ब) 8 बिट (क) 4 बाइट (ड) 8 बाइट असतात.

प्रश्न 3.3 सिंगल फ्लिप-फ्लॉप _____ बिट्स माहिती सेटयित करू शकते.

- (अ) 2 (ब) 32 (क) 64 (ड) 1

प्रश्न 3.4 8086 मायक्रोप्रोसेसरमध्ये किती पिन आहेत?

- (अ) 20 (ब) 60 (क) 40 (ड) 30

प्रश्न 3.5.1 MB मेमोरी समतुल्य

- (अ) 1024 बिट्स (ब) 1024 KB (क) 1024 बाइट्स (ड) 1024 GB

प्रश्न 3.6: खालीलपैकी कोणते योग्य आहे?

- (अ) मायक्रोप्रोसेसरमध्ये एएलयू, फ्लॅश मेमोरी आणि कंट्रोल एकक असतात.
 (ब) मायक्रोप्रोसेसरमध्ये एएलयू, रजिस्टर्स आणि कंट्रोल युनिट्स असतात.
 (क) मायक्रोकंट्रोलरमध्ये फक्त एएलयू आणि कंट्रोल युनिट्स असतात
 (ड) मायक्रोप्रोसेसरमध्ये फक्त एएलयू असते.

प्रश्न 3.7 8086 मायक्रोप्रोसेसरमध्ये, ॲड्रेस बस

- (अ) एकाच वेळी 16 बिट्स मोजते. (ब) एकाच वेळी दोन 8-बिट व्हॅल्यू मिळवू शकते.
 (क) 20 ॲड्रेस लाईन्स असतात. (ड) वरीलपैकी काहीही नाही.

प्रश्न 3.8 अंकगणिताच्या आणि तर्कशास्त्राच्या कार्याच्या परिणामांच्या स्वरूपाविषयी माहिती साठवणाऱ्या रजिस्टरला --- असे म्हणतात.

- (अ) फ्लॅग रजिस्टर (ब) संचायक (क) प्रोग्राम काउंटर (ड) प्रक्रिया स्थिती रजिस्टर

प्रश्न 3.9 नेमोनिक्सचा वापर करणाऱ्या प्रोग्रामला

- (अ) ऑब्जेक्ट प्रोग्राम (ब) फॅच सायकल (क) असेंब्ली लॅंग्वेज (ड) मायक्रो इंस्ट्रक्शन म्हणतात.

प्रश्न 3.10 खालील 8086 इंस्ट्रक्शनचे अॅड्रेसिंग मोड ओळखा.

ADD R1,R2

(अ) डायरेक्ट (ब) इमेडिएट (क) इम्प्लईड (ड) रजिस्टर

प्रश्न 3.11 8086 मायक्रोप्रोसेसरमध्ये, प्रोग्राम स्टेटस वर्ड रजिस्टर जोडी खालीलपैकी कोणती रजिस्टर जोडी म्हणून अंमलात आणली जाते?

(अ) प्रोग्राम काउंटर आणि स्टॅक पॉइंटर (ब) प्रोग्राम काउंटर आणि संचायक
(क) प्रोग्राम काउंटर आणि ध्वज रजिस्टर (ड) स्थिती ध्वज आणि कंट्रोल ध्वज

प्रश्न 3.12 8086 मायक्रोप्रोसेसरच्या स्टॅकबद्दल खालीलपैकी कोणते खरे नाही?

(अ) स्टॅक ही शेवटची पहिली रचना आहे
(ब) जेव्हा तुम्ही स्टॅकवर जोर देता, तेव्हा तेथे माहिती जतन केली जाते.
(क) स्टॅकसाठीचा रजिस्टर 8 बिट्सचा बनलेला असतो.
(ड) स्टॅक बंद करून त्यावर माहिती पुनर्प्राप्त केली जाते

प्रश्न 3.13 8086 मायक्रोप्रोसेसरमध्ये, कंडिशनल ब्रांच स्टेटमेंट खालीलपैकी कोणतेही फ्लॅग बदलत नाही.

(अ) झीरो फ्लॅग (ब) कॅरी फ्लॅग (क) साइन फ्लॅग (ड) कोणताही पर्याय नाही

प्रश्न 3.14 _____ अॅड्रेसिंग मोड ऑपरेंडचा प्रभावी ऍड्रेस मिळविण्यासाठी ऑफसेट आणि इंडेक्स रजिस्टर जोडतो.

(अ) इंडेक्स (ब) बेस-इंडेक्स (क) रजिस्टर इंडारेक्ट (ड) रिलॅटिव्ह

बहुपर्यायी प्रश्नाचे उत्तरे

3.1	(अ)	3.2	(ड)	3.3	(ड)	3.4	(क)	3.5	(ब)	3.6	(ब)
3.7	(क)	3.8	(अ)	3.9	(क)	3.10	(ड)	3.11	(ड)	3.12	(क)
3.13	(ड)	3.14	(अ)								

लघु आणि दीर्घ उत्तर प्रकार प्रश्न

श्रेणी-I

प्रश्न 3.1 इनपुट आणि आउटपुट पोर्ट म्हणजे काय?

प्रश्न 3.2 मायक्रोप्रोसेसर, मायक्रोकंट्रोलर आणि मायक्रोकंप्यूटर यात फरक करा?

प्रश्न 3.3 AD7-0 पिनचे महत्त्व काय आहे?

प्रश्न 3.4 प्रोसेसरवर प्रोग्रामरचा दृष्टीकोन म्हणजे काय?

प्रश्न 3.5 रजिस्टर म्हणजे काय आणि इतर सामान्य उद्देश रजिस्टरपेक्षा त्याचे फायदे आणि तोटे काय आहेत?

प्रश्न 3.6: इतर सामान्य उद्देश रजिस्टरपासून रजिस्टर वेगळे काय आहे?

प्रश्न 3.7 'AX', 'BX', 'CX' आणि 'DX' रेजिस्टर्स कशा वापरल्या जाऊ शकतात याचे वर्णन करा?

प्रश्न 3.8 HL जोडीला इतर रेजिस्टर जोड्यांपासून काय वेगळे करते?

प्रश्न 3.9 वापरकर्ते सहसा हेक्साडेसिमल नोटेशनमध्ये अॅड्रेसिंग आणि डेटा का निर्दिष्ट करतात?

प्रश्न 3.10 मायक्रोप्रोसेसरच्या विकासाचे थोडक्यात वर्णन करा?

प्रश्न 3.11 8086 मधील पुश आणि पॉप इन्स्ट्रक्शन समजावून सांगा?

प्रश्न 3.12 8088 मायक्रोप्रोसेसर हा 16-बिट ऐवजी 8-बिट मायक्रोप्रोसेसर का आहे?

श्रेणी-II

प्रश्न 3.13 मायक्रोप्रोसेसर मायक्रोप्रोग्रामपेक्षा कसा वेगळा आहे? मायक्रोप्रोग्रामच्या वापराशिवाय मायक्रोप्रोसेसर तयार केला जाऊ शकतो का? सर्व मायक्रोप्रोग्राम केलेले संगणक मायक्रोप्रोसेसर म्हणूनही काम करतात हे खरे आहे का?

प्रश्न 3.14 इन्स्ट्रक्शन सेट आर्किटेक्चर म्हणजे काय? आरआयएससी आणि सीआयएससी इन्स्ट्रक्शन सेट आर्किटेक्चरमधील फरक स्पष्ट करा? आरआयएससी किंवा सीआयएससी वापरत असलेल्या मायक्रोप्रोसेसर किंवा प्रोसेसरचे नाव द्या.

प्रश्न 3.15 इन्स्ट्रक्शनचे विविध स्वरूप समजावून सांगा. विशिष्ट मायक्रोप्रोसेसरसाठी कोणते डारेक्टिव स्वरूप योग्य आहे हे कसे ठरवायचे?

प्रश्न 3.16 मायक्रोप्रोसेसरमध्ये एड्रेसिंग मोडची आवश्यकता काय आहे? विविध अडुरेसींग पद्धती आणि त्यांचे अनुप्रयोग तपशीलवार समजावून सांगा.

प्रश्न 3.17 8086 मायक्रोप्रोसेसरचे कार्यात्मक भाग तपशीलवार समजावून सांगा.

संख्यात्मक समस्या:

प्रश्न 3.18 रजिस्टर R3 आणि R4 मध्ये अनुक्रमे 300 आणि 1200 दशांश व्हॅल्यू असतात आणि प्रोसेसरची वर्ड लांबी 32 बिट्स असते. दशांश मध्ये "STORE R5,90 (R3, R4)" डारेक्टिवचा प्रभावी ऍड्रेस _____ असेल.

[Ans: 1590]

प्रश्न 3.19 दोन शब्दांची इन्स्ट्रक्शन 'A' या ऍड्रेसवर ठेवली जाते आणि 'B' हे डारेक्टिवचे ऍड्रेस फील्ड दर्शवते, जे 'A + 1' येथे आहे. इन्स्ट्रक्शन अंमलात आणताना वापरलेला ऑपरेंड 'C' या ऍड्रेसवर संग्रहित केला जातो. डारेक्टिवक रजिस्टरमध्ये, X हे व्हॅल्यू साठवले जाते. जर शिक्षणाची अडुरेसींग पद्धती (i) अप्रत्यक्ष, (ii) थेट, (iii) अनुक्रमित आणि (iv) सापेक्ष असेल तर इतर पत्त्यांवरून 'C' ची गणना कशी केली जाते ते स्पष्ट करा.

[उत्तर: (i) $C = M[B]$ (ii) $C = B$ (iii) $C = B + X$ (iv) $C = B + A + 2$]

प्रश्न 3.20 मेमोरी अॅड्रेस 600 वर एक ब्रांच इन्स्ट्रक्शन आहे. शाखेचा ऍड्रेस 300 दशांश ऍड्रेसवर निर्दिष्ट केला आहे.

(i) दशांश आणि द्वैती संख्यांमध्ये सापेक्ष ऍड्रेस फिक्स्ड करा.

(ii) आणण्याच्या टप्प्यानंतर संगणकात '300' चे द्वैती प्रतिनिधित्व शोधा. मग दाखवा की बायनरीमध्ये 300 हे प्रोग्राम काउंटर (पी. सी.) च्या बेरीज आणि त्यात आढळणाऱ्या सापेक्ष एंड्रेसच्या बरोबरीचे आहे. (i).

[उत्तर: (i) सापेक्ष एंड्रेस = $300 - 601 = -301$; $301 = 000100101101$; $-301 = 111011010011$ (ii) $PC = 601 = 001001011001$, $300 = 000100101100$; $PC = 001001011001$ (601) $RA = 111011010011$ (-301) $EA = 300 = 000100101100$].

प्रश्न 3.21 जर इंस्ट्रक्शन (i) संगणकीय असेल आणि मेमोरीमधून ऑपरेंडची आवश्यकता असेल, किंवा (ii) ब्रांच असेल, तर अप्रत्यक्ष अडुरेसींग मोडमध्ये इंस्ट्रक्शन आणण्यासाठी आणि अंमलात आणण्यासाठी कंट्रोल युनिटद्वारे केलेल्या मेमोरी प्रवेशांची संख्या फिक्स्ड करा.

[उत्तर: (i) 3 (ii) 2]

प्रश्न 3.22 इंडेक्स्ड अॅड्रेसिंग मोडमध्ये, इंस्ट्रक्शनच्या एंड्रेस फील्डमध्ये रजिस्टर इन्डायरेक्ट मोड इंस्ट्रक्शनच्या समतुल्य मानण्यासाठी कोणते व्हॅल्यू प्रविष्ट केले पाहिजे?

[उत्तर: 0]

प्रश्न 3.23 एंड्रेस क्षेत्राचे व्हॅल्यू 400 आहे. प्रोसेसरच्या R1 रजिस्टरमध्ये 200 हा क्रमांक असतो. जर इंस्ट्रक्शन संबोधित करण्याची पद्धत (i) तात्काळ (ii) थेट (iii) अप्रत्यक्ष रजिस्टर (iv) सापेक्ष किंवा (v) R1 डारेक्टिवक रजिस्टरसह डारेक्टिवक असेल तर प्रभावी एंड्रेसचे व्हॅल्यूकन करा.

[उत्तर: (i) 501 (ii) 400 (iii) 200 (iv) $502 + 400 = 902$ (v) $200 + 400 = 600$]

प्रश्न 3.24 16 बिट्सची व्हॅल्यू 10011010110010101 आहे. कोणत्या ऑपरेशनची आवश्यकता आहे: (i) पहिल्या चार बिट्स, i.e., b0 ते b3, 0 पर्यंत साफ करा?

(ii) शेवटचे चार बिट्स, i.e., b12 ते b15, 1 वर सेट करा?

(iii) मध्य चार बिट्स, b6 ते b9 ला पूरक?

[उत्तर: (i) आणि 11111111110000 सह किंवा आणि 11111111111010 सह (ii) किंवा

1111000000000000 सह किंवा 0110000000000000 सह (iii) 000000111100000000 सह XOR]

प्रश्न 3.25 खाली दिलेल्या साइन केलेल्या प्रत्येक पूर्णांकाचे बायनरी प्रतिनिधित्व तयार करा.

(i) प्रथम, संख्या (+68) आणि (-83) ची बायनरी जोडणी करा आणि नंतर या ऑपरेशनच्या परिणामाचे व्हॅल्यूकन करा.

(ii) द्वैती संख्यांची (-68)-(+ 83) वजाबाकी करा आणि ओव्हरफ्लोची घटना फिक्स्ड करा.

(iii) बायनरी -68 चे व्हॅल्यू एका स्थानाने उजवीकडे सरकवा आणि परिणाम दशांश स्वरूपात दर्शवा. [संकेत: एरिथमेटिक शिफ्ट बरोबर करा कारण ही साइन केलेली संख्या आहे]

(iv) बायनरी -83 एक स्थान डावीकडे सरकवा आणि नंतर ओव्हरफ्लोची घटना फिक्स्ड करा.

[उत्तर: +83 = 01010011, +68 = 01000100, -83 = 10101101, -68 = 10111100 (i) -83 (10101101) + 68 (01000100) = -15 (11110001) (2 च्या पूरकामध्ये) (ii) -68 (10111100) -83 (10101101) = -151 (01101001) ओव्हरफ्लो (iii) -34 = 11011110, (iv) -166 ≠ 01011010, ओव्हरफ्लो]

प्रश्न 3.26 खालील इंस्ट्रक्शननुसार स्थिती बिट व्हॅल्यू C, S, Z आणि O फिक्स्ड करा. प्रत्येक उदाहरणामध्ये, रजिस्टर R 72 (हेक्साडेसिमल) संख्येपासून सुरू होते आणि ते 8-बिट रजिस्टर आहे.

(i) ADD C6, R (ii) ADD 1E, R (iii) SUB 9A, R

(iv) AND 8D, R (v) XOR R, R

[उत्तर: 138 (00111000) C = 1, S = 0, Z = 0, O = 0 (ii) 90 (10010000) C = 0, S = 1, Z = 0, O = 1 (iii) D8 (11011000) C = 0, S = 1, Z = 0, O = 1 (iv) 00 (00000000) C = 0, S = 0, Z = 1, O = 0 (v) 00 (00000000) C = 0, S = 0, Z = 1, O = 0]

प्रश्न 3.27: रजिस्टर A = 01010101 आणि B = 10101010 ची 8-बिट व्हॅल्यू विचारात घ्या.

(i) A आणि B या रजिस्टर मध्ये साठवलेल्या बायनरी व्हॅल्यूची जोडणी, त्या साइन संख्या आहेत असे गृहीत धरून करा.

(ii) (अ) मध्ये प्राप्त परिणाम (i) अनसाइन आणि

(ii) साइन केलेला आहे या गृहितकांनुसार दशांश समतुल्य प्रदान करा.

(iii) बेरीज केल्यानंतर स्थिती बिट व्हॅल्यू C, S, Z आणि O फिक्स्ड करा.

[उत्तर: (i) A + B = 11111111 (ii) अनसाइन = 255, साइन = -1 (iii) C = 0, S = 1, Z = 0, O = 0]

प्रश्न 3.28 A = 01000001 आणि B = 10000101 या दोन अनसाइन संख्यांची तुलना संगणक प्रोग्रामद्वारे केली जाते. वजाबाकीचा (A-B) बायनरी परिणाम आणि स्थिती बिट्स सी आणि झेडची व्हॅल्यू फिक्स्ड करा.

[उत्तर: A-B = 10111100, C = 1; Z = 0]

प्रश्न 3.29 A = 01000001 आणि B = 10000100 या दोन अनसाइन संख्यांची तुलना संगणक प्रोग्रामद्वारे केली जाते. वजाबाकीचा (A-B) बायनरी परिणाम आणि स्थिती बिट्स S, Z आणि O ची व्हॅल्यू फिक्स्ड करा.

[उत्तर: A-B = + 65 (01000001)-- 124 (10000100) = 189 (10111101) = 010111101 (9 बिट्स) S = 1, Z = 0, O = 1 (ओव्हरफ्लो)]

प्रश्न 3.30: स्टॅक पॉइंटरचे व्हॅल्यू 3560 आहे, तर मेमोरी स्टॅकच्या शीर्षस्थानी 5320 आहे. दोन शब्दांची कॉल सबरूटीन इंस्ट्रक्शन 1120 मध्ये आहे आणि त्यानंतर स्थान 1121 वर 6720 चे ऍड्रेस फील्ड आहे. PC, SP आणि स्टॅकच्या वरच्या भागाची सामग्री काय आहे:

- (i) मेमोरीतून कॉल इंस्ट्रक्शन घेण्यापूर्वी?
- (ii) कॉल इंस्ट्रक्शन अंमलात आणल्यानंतर?
- (iii) सबरूटीन परत आल्यानंतर?

[उत्तर: पीसी: 1120 (प्रारंभिक) 6720 (कॉल केल्यानंतर) 1122 (after return) (ii) एसपी: 3560 (प्रारंभिक) 3559 (कॉल केल्यानंतर) 3560 (after return) (iii) स्टॅकचा वरचा भाग 5320 (प्रारंभिक) 1122 (कॉल केल्यानंतर) 5320 (परतल्यानंतर)]

प्रात्यक्षिक:

उद्देश: 16-बिट संख्यांचा वापर करून बेरीज आणि वजाबाकी करा. सुरुवातीला असे समजा की दोन्ही संख्या मेमोरीमध्ये राहत आहेत. तुम्हाला ही संख्या CPU रजिस्टरमध्ये लोड करावी लागेल, या संख्यांची बेरीज आणि वजाबाकी करावी लागेल आणि निकाल परत मेमोरी मध्ये संग्रहित करावा लागेल.

साधने: 8086 मायक्रोप्रोसेसर सेट [3], वीज पुरवठा

सिद्धांत: ADD इंस्ट्रक्शन बेरीज करते, तर SUB इंस्ट्रक्शन वजाबाकी करते.

ADD इंस्ट्रक्शन डेस्टिनेशन रजिस्टर किंवा मेमोरी स्थानाच्या सामग्रीमध्ये निर्देशात निर्दिष्ट केलेला तात्काळ डेटा किंवा मेमोरी स्थान किंवा सोर्स रजिस्टरची सामग्री जोडू शकते.

परिणाम डेस्टिनेशन ऑपरेंडमध्ये संग्रहित केला जातो. तथापि, सोर्स आणि डेस्टिनेशन ऑपरेंड हे दोन्ही मेमोरी ऑपरेंड असू शकत नाहीत. कारण मेमोरीमध्ये मेमोरी जोडणे शक्य नाही.

सर्व कंडिशन कोड फ्लॅग परिणामाने प्रभावित होतात. याव्यतिरिक्त, कॅरी फ्लॅगची स्थिती तपासणे आवश्यक आहे आणि निकाल आणि कॅरी फ्लॅग दोन्ही मेमोरी स्थानावर जतन केले जातात. त्याचप्रमाणे, वजाबाकी ऑपरेशन कार्यान्वित करा आणि परिणाम योग्य आहे याची खात्री करण्यासाठी ओव्हरफ्लो स्थिती तपासा.

मायक्रोप्रोसेसर खालीलप्रमाणे विशिष्ट आदेशांच्या मदतीने कार्य करू शकतो.

आरंभ ऍड्रेस प्रविष्ट करा पुनर्संचयित करा

निर्देशाद्वारे प्रोग्राम इंस्ट्रक्शन भरा मेमोरीमध्ये निकाल सेटयित करा कार्यान्वित करा.

16-बिट जोडणीची प्रक्रिया:

1. प्रथम तुम्हाला असेंब्ली लँग्वेज प्रोग्राम लिहावा लागेल, एक प्रोग्राम एडिशन करण्यासाठी आणि दुसरा प्रोग्राम सबट्रॅक्शन करण्यासाठी लिहावा लागेल.
2. तुम्हाला मेमोरी ऍड्रेस, मशीन ऑपकोड, नेमोनिक्स, ऑपरेंड म्हणून स्तंभाचे नाव असलेले टेबल तयार करावे लागेल.
3. प्रोग्राम सुरू करा.
4. पहिला डेटा AX रजिस्टरमध्ये आणि दुसरा डेटा BX रजिस्टरमध्ये लोड करा.
5. वाहून नेण्यासाठी CL रजिस्टर साफ करा.
6. AX आणि BX रजिस्टर्समध्ये ऑपरेशन जोडा आणि AX रजिस्टरमध्ये निकाल मिळवा.
7. निकाल मेमोरी ऍड्रेसवर सेटयित करा.



स्कॅन करा

8086 मायक्रोप्रोसेसर किटचे कार्य समजून घेण्यासाठी



स्कॅन करा

8086 मायक्रोप्रोसेसर किटवर वजा करण्यासाठी



स्कॅन करा

8086 मायक्रोप्रोसेसर किटवर अतिरिक्त कामगिरी करण्यासाठी

8. जर कॅरी = 1 असेल तर पुढील स्टेपवर जा. अन्यथा, 11 व्या स्टेपवर जा.
9. कॅरी इन मेमोरी वाढवा.
10. कॅरी मेमोरीमध्ये ठेवा.
11. प्रोग्राम थांबवा.

अधिक जाणून घ्या:

भारतीयांनी केलेले नवकल्पना



शक्ती हा भारतातील पहिला मुक्त सोर्स औद्योगिक दर्जाचा प्रोसेसर आहे [4]. प्राध्यापक व्ही. कामकोटी यांच्या देखरेखीखाली आयआयटी मद्रास येथील 'रिकॉन्फिगरेबल इंटेलिजेन्ट सिस्टीम्स इंजिनिअरिंग' समूहाने हे विकसित केले आहे. जागा, विजेचा वापर आणि कामगिरीच्या बाबतीत व्यावसायिक उत्पादनांशी स्पर्धात्मक असलेले एस. ओ. सी. विकसित करणे हा या चमूचा उद्देश आहे. सर्व शक्ती स्रोत कोड ओपनसोर्स आहेत. शक्ती प्रोसेसर एम्बेडेड अनुप्रयोग, रोबोटिक नियंत्रक आणि इंटरनेट ऑफ थिंग्स बोर्डसाठी आहेत [7].

पुनरुज्जीवित केलेल्या पारंपरिक भारतीय मातीच्या स्वयंपाकाच्या वस्तू वापरण्याचे आरोग्यविषयक फायदे

भारतीय म्हण "आम्ही पोटाद्वारे इतर लोकांच्या हृदयापर्यंत पोहोचतो" [5]. अन्नाचा आपल्यावर मानसिक परिणामही होतो. स्वयंपाकासाठी घटक मिसळणे, गरम करणे आणि एकत्र करणे आवश्यक आहे. स्वयंपाकासाठीची भांडी आणि गरम करणारे घटक (कोळसा, मातीचा स्टोव, केरोसीन स्टोव, गॅस स्टोव, मायक्रोवेव्ह) गरम करण्याचे घटक बनवतात. पूर्वज कास्ट, तांबे, पितळ, लोह आणि मातीची भांडी वापरून शिजवतात.



ही भांडी अन्नाचे पौष्टिक व्हॅल्यू अबाधित ठेवतात [6]. आधुनिक प्रेशर कुकर, अॅल्युमिनियम, पोलाद, प्लास्टिक आणि नॉनस्टिक यासारखी आधुनिक भांडी अन्नामध्ये आढळणारी बहुतांश सूक्ष्म पोषक तत्त्वे राखू शकत नाहीत. खरं तर, प्लास्टिक, अॅल्युमिनियम आणि नॉनस्टिक भांडी प्रत्यक्षात अन्नामध्ये काही विषारी पदार्थ वाढवतात. उदाहरणार्थ, कास्ट आयर्नची भांडी अन्नातील लोहाचे प्रमाण वाढवण्यात, विशेषतः अम्लीय पदार्थांच्या बाबतीत, महत्त्वपूर्ण भूमिका बजावतात. तांब्यामध्ये आहारातील कोलेजन वर्धक बळकट करण्याची विशेष क्षमता असते. पाणी जतन करण्यासाठी पितळ उत्कृष्ट आहे, ज्यामुळे मानवी रोगप्रतिकारक शक्ती वाढते. त्याचप्रमाणे, मनःस्थितीच्या आजारावर चांदीचा शांत प्रभाव असतो, परंतु ती प्रतिबंधितपणे महागडी आणि मिळवणे कठीण असते. महत्त्वाचे म्हणजे, मातीची भांडी रोगप्रतिकारक शक्तीवर लक्षणीय परिणाम करतात. जेव्हा जेव्हा आपण मातीच्या भांड्यांचा विचार करतो, तेव्हा आपण आपल्या जवळजवळ नामशेष झालेल्या मातीच्या कुंभारांचा विचार करू शकतो. मातीच्या कपांऐवजी, चहा विक्रेते आता एकतर बिस्फेनॉल ए (बी. पी. ए.) असलेले

प्लास्टिकचे कप किंवा कागदाचे कप (स्टायरीन असलेले) वापरतात, जे दोन्ही मानवी आरोग्यासाठी अत्यंत हानिकारक आहेत.

मातीची भांडी या कारणांमुळे आपल्याला चिंतेत टाकतात. चिकणमातीमध्ये शिजवलेले अन्न अधिक पोषकद्रव्ये राखून ठेवते आणि त्याची चव अधिक चांगली असते. मातीची भांडी रोजगार निर्माण करतात. एकाच वेळी आरोग्यदायी आणि अधिक स्वादिष्ट अशी जीवनशैली विकसित करण्यासाठी, मातीची भांडी आणि भारतातील आश्चर्यकारक खाद्यपदार्थांमध्ये स्वयंपाक करण्यास प्रोत्साहन देणे महत्वाचे आहे. मातीची भांडी स्वयंपाकासाठी कमी पाणी वापरतात आणि कचरा जागा घेत नाहीत. भारतीय मातीमध्ये खोलवर रुजलेले आहेत, आता ही माती देशभरात आणि देशाबाहेर चव आणि आरोग्य वितरीत करण्यासाठी काही उपजीविकेसाठी वापरण्याची वेळ आली आहे.

संदर्भ आणि सुचवलेले वाचन

[1] एम. मॉरिस मनो, संगणक प्रणाली वास्तुकला. प्रेंटिस-हॉल, इंक., तिसरी आवृत्ती.

<https://poojavaishnav.files.wordpress.com/2015/05/mano-m-computer-system-Architecture.pdf> (last accessed: May 2023)

[2] कार्ल हमाकर, इवोन्को वॅनेसिक, सफवत झाकी आणि नरैग मंजिकियन, संगणक संस्था आणि एम्बेडेड सिस्टीम्स. माकग्रा-हिल उच्च शिक्षण, 2011.

[3] मायक्रोप्रोसेसर आणि इंटरफेसिंग लॅब मान्युअल

https://webstor.srmist.edu.in/web_assets/srm_mainsite/files/2017/cselab-manual-microprocessor.pdf (last accessed: May 2023)

[4] शक्ती प्रोसेसर. <https://shakti.org.in/> (last accessed: May 2023)

[5] देबदीप खान, सुदत्ता बॅनर्जी, प्राचीन भारतीय मातीची भांडी पुनरुज्जीवित करणे आणि आरोग्यावर त्याचा प्रभाव. इंटरनॅशनल जर्नल ऑफ ऑल रिसर्च एज्युकेशन अँड सायन्टिफिक मेथड्स (IJARESM) ISSN: 2455-6211, ब्लॉक 8, अंक 7, जुलै 2020.

[6] दीर्घायुष्यासाठी वैदिक स्वयंपाक. <https://vedichindustan.org/vedic-cooking Long-life/> (last accessed: May 2023)

[7] शांतनु चट्टोपाध्याय, मायक्रोप्रोसेसर आणि मायक्रोकंट्रोलर्स, आयआयटी खरगपूर, 2023 द्वारे एनपीटीईएल अभ्यासक्रम.

<https://nptel.ac.in/cours/108105102> वर संपर्क साधा. (last accessed: May 2023)



स्कॅन करा
8085 पिन आकृतीसाठी



स्कॅन करा
8086 पिन आकृतीसाठी

4. असेम्बली लॉगवेज प्रोग्रामिंग

युनिट ची वैशिष्ट्ये

या युनिटमध्ये खालील पैलूवर चर्चा केली आहे:

- असेम्बली भाषेची सामान्य संरचना;
- असेम्बलर डारेक्टिव्ह, कार्यपद्धती आणि माक्रो;
- एरिथमेटिक, लॉजिक, ब्रान्च आणि कॉल इंस्ट्रक्शनचा समावेश असलेले असेम्बली भाषेचे प्रोग्राम;
- एरिथमेटिक अभिव्यक्ती, स्ट्रिंग मानिपुलेशन आणि सॉर्टिंगचे व्हॅल्यूकन करण्यासाठी प्रोग्राम.

अधिक जिज्ञासा आणि सर्जनशीलता वाढवण्यासाठी आणि समस्या सोडवण्याची कौशल्ये वाढवण्याच्या उद्देशाने विषयांचे व्यावहारिक अनुप्रयोग सादर केले जातात. ब्लूम वर्गीकरणशास्त्राच्या खालच्या आणि वरच्या स्तरांनुसार दोन श्रेणींमध्ये चिन्हांकित केलेल्या मोठ्या संख्येने बहुपर्यायी प्रश्न आणि लहान किंवा दीर्घ उत्तराच्या प्रश्नांव्यतिरिक्त, युनिट संख्यात्मक समस्यांच्या स्वरूपात सराव असाइनमेंट, संदर्भाची यादी आणि सुचविलेले वाचन प्रदान करते. हे लक्षात घेणे महत्वाचे आहे की स्वारस्य असलेल्या विविध विषयांवरील अधिक माहितीसाठी स्कॅन केले जाऊ शकणारे अनेक क्यूआर कोड वेगवेगळ्या भागांमध्ये समाविष्ट केले गेले आहेत आणि आवश्यक आधार डेटा प्राप्त करण्यासाठी वापरले जाऊ शकतात.

सामग्रीवर आधारित संबंधित व्यावहारिक नंतर विषयावरील "अधिक जाणून घ्या" विभाग आहे. हा विभाग काळजीपूर्वक अशा प्रकारे तयार करण्यात आला आहे की त्यात असलेली पूरक माहिती पुस्तकाच्या वाचकांसाठी मौल्यवान असेल. हा विभाग प्रामुख्याने संगणक प्रणाली संघटनेच्या विकासासाठी आणि डारेक्टिव सेट संरचनेच्या इतिहासासाठी भारतीय नवसंशोधकांच्या योगदानावर लक्ष केंद्रित करतो. चांगले आरोग्य प्राप्त करण्यासाठी, मन-शरीर चांगले करण्यासाठी योग आणि प्राणायाम या भारतीय पद्धतींची चर्चा देखील केली गेली आहे.

तर्क

हा अध्याय ओपन सोर्स NASM असेम्बली प्रोग्रामिंग लॉगवेज वापर करून असेम्बली भाषेमागील मूलभूत कल्पना, तसेच त्याची रचना, सामान्य संकल्पना आणि टर्मिनॉलॉजी तपासतो. हार्डवेअर डेव्हलपर्सच्या दृष्टिकोनातून असेम्बली लॉगवेज आवश्यकतेवर चर्चा केली जाते. असेम्बली भाषांच्या सामान्य वर्ड्थार्चे परीक्षण असेम्बली भाषेत सोप्या ते गुंतागुंतीच्या प्रोग्राम्स लिहिण्याचा पाया घालते. हा अध्याय असेम्बली भाषेचे प्रोग्राम लिहिण्यासाठी विविध प्रकारच्या इंस्ट्रक्शनचा वापर स्पष्ट करतो. एरिथमेटिक, लॉजिक, ब्रान्च आणि कॉल इंस्ट्रक्शनचा समावेश असलेल्या असेम्बली प्रोग्रामसारख्या मूलभूत प्रोग्रामपासून प्रगत प्रोग्रामपर्यंत, एरिथमेटिक एक्सप्रेसनचे व्हॅल्यूकन करणे, स्ट्रिंग हाताळणी आणि क्रमवारी लावणे यावर या अध्यायात चर्चा केली आहे. हा अध्याय परिचय म्हणून काम करतो आणि डारेक्टिव सेट आणि असेम्बली भाषांच्या अधिक सखोल अभ्यासाचा पाया घालतो.

पूर्व-आवश्यकता

मायक्रोप्रोसेसर आर्किटेक्चर (Unit-III)

डिजिटल इलेक्ट्रॉनिक्स: संख्या प्रणाली, बायनरी ऑक्टल आणि हेक्साडेसिमल (Polytechnic Engineering)

ऑपरेटिंग सिस्टिम: प्रणाली कॉल, लिनक्स ऑपरेटिंग सिस्टिम चे कर्नल इत्यादींची मूलभूत तत्त्वे.

युनिट निष्पत्ती

या युनिटच्या परिणामांची यादी खालीलप्रमाणे आहे:

U4-O1: प्रोसेसर डिझाइनमध्ये असेम्बली लॉगवेज प्रोग्रामिंगच्या भूमिकेचे वर्णन करा.

U4-O2: साध्या असेम्बली प्रोग्राम लिहिण्याच्या मुख्य वैशिष्ट्यांचे वर्णन करा.

U4-O3: असेम्बलर डारेक्टिव स्पष्ट करा.

U4-O4: प्रोसीजर आणि माक्रोचे महत्त्व स्पष्ट करा.

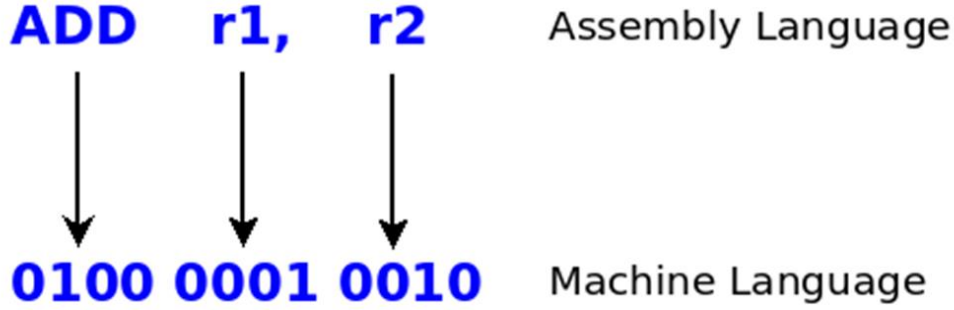
U4-O5: जटिल असेम्बली प्रोग्राम लिहिण्याची रचना तत्त्वे.

युनिट -4 निष्पत्ती	निष्पत्ती सह अपेक्षित मापिंग				
	3- कमकुवत मापिंग, 2- मध्यम मापिंग, 3- मजबूत मापिंग				
	CO-1	CO-2	CO-3	CO-4	CO-5
U4-01	3	3	3	1	2
U4-02	3	1	2	1	2
U4-03	2	2	3	1	2
U4-04	3	3	3	3	2
U4-05	3	3	3	2	3

4.1 परिचय

असेम्बली लॉगवेज प्रोग्राम विकास सुलभ करते. प्रोग्रामर संगणक प्रणालींच्या हार्डवेअर साठी प्रोग्राम लिहू शकतो. असेम्बली लॉगवेज ही इन्स्ट्रक्शन सेट आर्किटेक्चर (ISA) आणि असेंबलर-स्पेसिफिक लो-लेव्हल प्रोग्रामिंग आहे. असेम्बली लॉगवेज कार्यक्रमांमध्ये असेम्बली विधाने असतात. प्रत्येक संगणक इन्स्ट्रक्शन ही 'ADD', 'SUB', 'LOAD' इ. सारख्या मजकूराची ओळख पटवणारी असते. असेम्बली भाषेचे प्रोग्राम सर्व प्रतीकात्मक नावे आणि त्यांचे नियम वापरतात. प्रत्येक इन्स्ट्रक्शनमध्ये त्यांच्या व्हॅल्यूसह किंवा ऍड्रेससह ऑपरेंडची यादी देखील समाविष्ट असते. ऑपरेंडची व्हॅल्यू एकतर संख्यात्मक असतात किंवा रजिस्टर किंवा मेमोरीमध्ये संग्रहित केली जातात.

असेम्बली भाषेचे प्रोग्राम असेम्बलर्सद्वारे मशीन इंस्ट्रक्शनमध्ये रूपांतरित केले जातात. असेम्बलर प्रोग्राम्ससारखे युटिलिटी प्रोग्राम्स संगणक प्रणाली सॉफ्टवेअरमध्ये समाविष्ट केले जातात. असेम्बलर्स, इतर सॉफ्टवेअरप्रमाणेच, संगणकाच्या मेमोरीमध्ये मशीन इंस्ट्रक्शन म्हणून संग्रहित केले जातात. मशीन इंस्ट्रक्शन बायनरी नमुन्यांनी बनलेल्या असतात. अशा संरचनांचा वापर करून प्रोग्राम करणे आव्हानात्मक आहे. म्हणून, प्रतीकात्मक नावे असेम्बली भाषेतील बायनरी कोडचे नमुने प्रतिबिंबित करतात.



आकृती. 4.1: मशीनी भाषेत असेम्बली लॉगवेज ADD डारेक्टिव प्रतिनिधित्व

प्रोग्रामर असेम्बलर द्वारे असेम्बली लॉगवेज प्रोग्राम तयार केल्यानंतर तो संगणकाद्वारे कार्यान्वित करण्या करीता मशीनी लॉगवेज प्रोग्राम रूपांतरित केला जातो. आकृती 4.1 मध्ये असेम्बली लॉगवेज इंस्ट्रक्शन, i.e., ADD r1, r2 आणि मशीन लॉगवेज इंस्ट्रक्शनचे उदाहरण दर्शविते जे त्यातून तयार केले जाऊ शकते. ADD इंस्ट्रक्शन रेजिस्टर r1 आणि r2 मध्ये साठवलेल्या व्हॅल्यूची भर घालते आणि परिणाम रेजिस्टर r1 मध्ये स्टोअर होतो.

असेम्बली लॉगवेज इंस्ट्रक्शन मानवी रिडेबल स्वरूपात आहेत आणि मशीन कोडचे मोहक लिखित प्रतिनिधित्व करतात. यामुळे प्रोग्राम लिहिणे खूप सोपे, सुबकपणे वापरले जाते. तथापि, प्रोग्रामिंग कंटाळवाणे आहे कारण प्रत्येक इंस्ट्रक्शनद्वारे कमी प्रमाणात काम केले जाते आणि प्रोग्रामरला उपलब्ध असलेल्या इंस्ट्रक्शन मशीननुसार भिन्न असतात.

जर एखाद्या प्रोग्रामरला वेगळ्या प्रकारच्या संगणकावर प्रोग्राम चालवायचा असेल, तर प्रोग्राम पूर्णपणे नवीन संगणकाच्या असेम्बली भाषेत पुन्हा लिहावा लागेल.

व्यवहारात, स्वतंत्र असेम्बली प्रोग्राम्स असेम्बलरचा वापर करून लिहिले जाऊ शकतात आणि एक्झिक्युटेबल मध्ये रूपांतरित केले जाऊ शकतात. C आणि C++ दोन्हीमध्ये असेम्बली कोडचे तुकडे समाविष्ट करण्याची क्षमता आहे. दुसरा पर्याय हा अधिक सामान्य पर्याय आहे. एकत्रित प्रोग्रामिंग नंतर कंपाइलरद्वारे मशीन कोडमध्ये रूपांतरित केले जाते. असेम्बली लॉगवेज चेअनेक फायदे असतात. असेम्बली कोड मशीन कोडप्रमाणेच अर्थपूर्ण आहे आणि याचे कारण असे आहे की प्रत्येक लाइन एक मशीन इंस्ट्रक्शन दर्शवते.

कीबोर्ड इनपुट वापरकर्त्याचे प्रोग्राम मेमोरीमध्ये किंवा हार्ड डिस्कवर संग्रहित करते. वापरकर्ता प्रोग्राम आता अल्फान्यूमेरिक वर्णांच्या ओळी आहेत. असेम्बलर प्रोग्राम वापरकर्त्याच्या प्रोग्रामचे विश्लेषण करतो आणि 0s आणि 1s च्या नमुन्यांसह मशीन-लॉगवेज प्रोग्राम तयार करतो जो संगणक कार्यान्वित करेल. सोर्स प्रोग्राम्स हे



स्कॅन करा

NASM ची तुलना

MASM. आणि GAS

असेम्बली भाषांशी करा

अल्फान्यूमेरिक मजकूर स्वरूपातील वापरकर्ता प्रोग्राम्स आहेत, तर ऑब्जेक्ट प्रोग्राम्स हे मशीन-लॅंगवेज चे प्रोग्राम्स आहेत. संगणकीय असेम्बली लॅंगवेज कॅपिटल आणि लोअर-केस अक्षरांमध्ये फरक करू शकते किंवा करू शकत नाही.

असेंबलर हार्ड ड्राइव्हवर ऑब्जेक्ट प्रोग्राम्स संचयीत करतो. ऑब्जेक्ट प्रोग्राम मुख्य मेमोरीमध्ये लोड करणे एक्सिक्युशन साठी आवश्यक आहे. यामुळे मेमोरीमध्ये लोडर युटिलिटी प्रोग्राम असणे आवश्यक आहे. जेव्हा लोडर चालवला जातो, तेव्हा तो हार्ड ड्राइव्हमधून मशीन-लॅंगवेज प्रोग्राम मेमोरीमध्ये पाठवतो. प्रोग्राम लांबी आणि मेमोरी ऍड्रेस लोडरला माहित असणे आवश्यक आहे. असेंबलर ही माहिती ऑब्जेक्ट कोडच्या आधी हेडरमध्ये ठेवतो. ऑब्जेक्ट कोड लोड केल्यानंतर, लोडर START सारख्या पहिल्या इंस्ट्रक्शनकडे वळतो. लोडरला रनटाइमवर वापरण्यासाठी असेम्बलरद्वारे ऑब्जेक्ट कोड हेडरमध्ये ऍड्रेस ठेवला जातो.



स्कॅन करा
विधानसभा भाषा
कार्यक्रमांसाठी ऑनलाईन
संकलक

या अध्यायात, असेम्बली प्रोग्रामच्या इंस्ट्रक्शन इंटेल 32 प्रोसेसरवर आधारित आहेत. एकाच निर्देशासाठी विविध मशीन एन्कोडिंग आहेत. असेम्बली प्रोग्रामिंगसाठी खालील असेंबलर्स मोठ्या प्रमाणावर वापरले जातात:

- मायक्रोसॉफ्ट असेंबलर (MASM)
- बोर्लँड टर्बो असेंबलर (TASM)
- जीएनयू असेंबलर (GAS)
- NASM असेंबलर

या अध्यायात, सर्व असेम्बली लॅंगवेज कार्यक्रमांवर NASM असेंबलर वापरून चर्चा केली आहे.

NASM हे लिनक्स आणि विंडोज या दोन्ही कार्यप्रणालीवर डाउनलोड करण्यासाठी विनामूल्य आहे. NASM हा इंटेल x86 आर्किटेक्चर असेंबलर आणि डिसएसेम्बलर आहे. हा निम्न-स्तरीय प्रोग्रामिंगसाठी एक लोकप्रिय पर्याय आहे आणि सामान्यतः ऑपरेटिंग सिस्टम, डिव्हाइस ड्रायव्हर्स आणि इतर प्रणाली-स्तरीय प्रोग्रामिंग क्रियाकलापांच्या डिझाइनमध्ये वापरला जातो. या अध्यायात, NASM चे मूलभूत वाक्यरचना आणि डारेक्टिव स्वरूप दर्शविले गेले आहेत जे चर्चा केलेल्या कार्यक्रमांसाठी वापरले जातात. NASM चांगले डॉक्यूमेंटेशन केलेले आहे. या अध्यायात चर्चा केलेल्या QR संकेतांचे आणि संदर्भांचे अनुसरण करून प्रगत प्रोग्राम तयार केले जाऊ शकतात.

4.2 असेम्बली लॉगवेज प्रोग्रामिंग

उबंटू ऑपरेटिंग सिस्टम डाउनलोड करण्यासाठी विनामूल्य आहे आणि त्याचे सतत समर्थन उपलब्ध आहे. तर उबंटू ऑपरेटिंग सिस्टमवर NASM असेंबलर



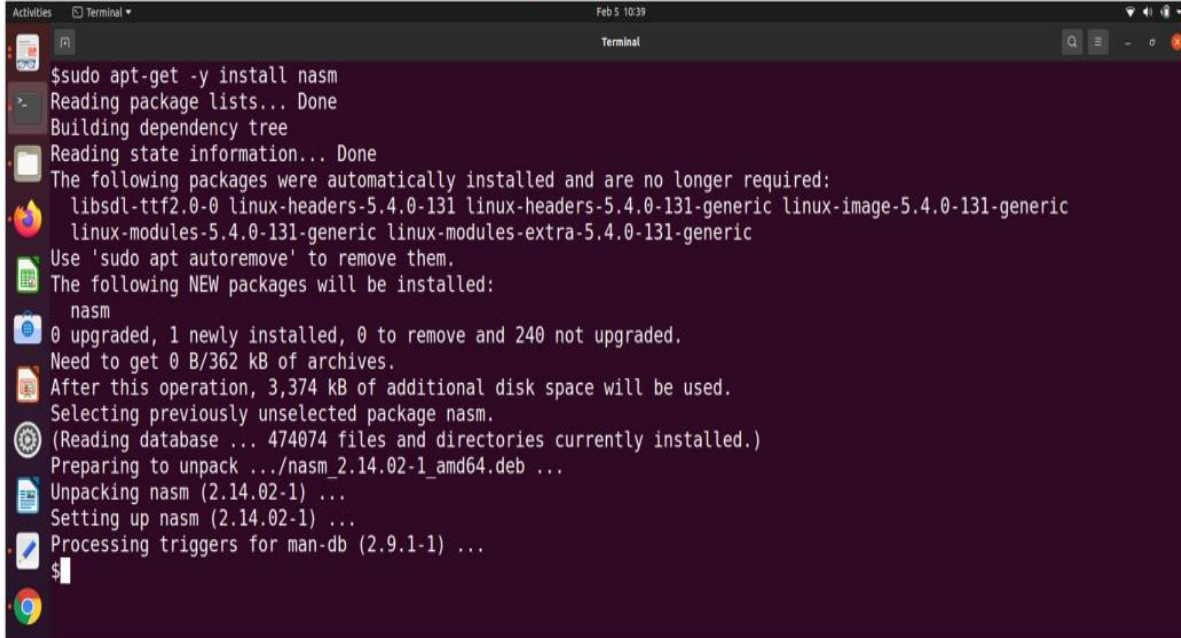
स्कॅन करा

NASM असेम्बली भाषा शिका

स्थापित आहे. NASM असेंबलरसह असेम्बली प्रोग्राम्स लिहिण्यास प्रारंभ करण्यासाठी खालील चरणांचे अनुसरण केले जाऊ शकते.

1. डेस्कटॉप किंवा लॅपटॉप प्रणालीवर 64 बिट उबंटू स्थापित करा (if your system is 64-bit)
2. त्यानंतर NASM स्थापित करण्यासाठी उबंटूच्या टर्मिनलमध्ये इंस्ट्रक्शन टाईप करा .


sudo apt-get -y install nasm



```
$sudo apt-get -y install nasm
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libstdl-ttf2.0-0 linux-headers-5.4.0-131 linux-headers-5.4.0-131-generic linux-image-5.4.0-131-generic
  linux-modules-5.4.0-131-generic linux-modules-extra-5.4.0-131-generic
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  nasm
0 upgraded, 1 newly installed, 0 to remove and 240 not upgraded.
Need to get 0 B/362 kB of archives.
After this operation, 3,374 kB of additional disk space will be used.
Selecting previously unselected package nasm.
(Reading database ... 474074 files and directories currently installed.)
Preparing to unpack .../nasm_2.14.02-1_amd64.deb ...
Unpacking nasm (2.14.02-1) ...
Setting up nasm (2.14.02-1) ...
Processing triggers for man-db (2.9.1-1) ...
$
```

3. प्रतिष्ठापनानंतर, NASM प्रतिष्ठापन स्थान खालील इंस्ट्रक्शन द्वारे तपासले जाऊ शकते.

whereis nasm



```
$whereis nasm
nasm: /usr/bin/nasm /usr/share/man/man1/nasm.1.gz
$
```

4. तुम्ही आज्ञेचा वापर करून स्थापित नास्मची आवृत्ती तपासू शकता.

nasm --version



```
$nasm --version
NASM version 2.14.02
$
```

5. तुम्हाला टर्मिनल स्क्रीन साफ करायची असल्यास, इंस्ट्रक्शन वापरा.

clear

```
Terminal
$ nasm --version
NASM version 2.14.02
$ clear
```

6. टर्मिनलची सध्याची कार्यरत निर्देशिका अशी तपासली जाऊ शकते.

pwd

```
Terminal
$ pwd
/home/user
$
```

7. आता तुम्ही एडिटर मध्ये असेम्बली लॉगवेज प्रोग्राम लिहिण्यास सुरुवात करू शकता. gedit संस्थापित आहे की नाही हे तपासण्यासाठी खालील इंस्ट्रक्शन टाईप करा.

sudo apt-get install gedit

```
Terminal
$ sudo apt-get install gedit
[sudo] password for user:
Reading package lists... Done
Building dependency tree
Reading state information... Done
gedit is already the newest version (3.36.2-0ubuntu1).
The following package was automatically installed and is no longer required:
  libstdc++2.0-0
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 247 not upgraded.
$
```

प्रतिष्ठापन सुरू करण्यासाठी प्रणाली संकेतवर्ड प्रविष्ट करा. जर ते आधीच इंस्टॉल केलेले असेल, तर तुम्हाला स्क्रीनशॉटमध्ये दाखवल्याप्रमाणे gedit आधीच इंस्टॉल केलेला आहे असा संदेश मिळेल. पुढील विभागात पहिला असेम्बली प्रोग्राम लिहिणे, त्याची वाक्यरचना आणि तो कसा चालवायचा याचा समावेश आहे.

```
Terminal
$ gedit print_name.asm
```

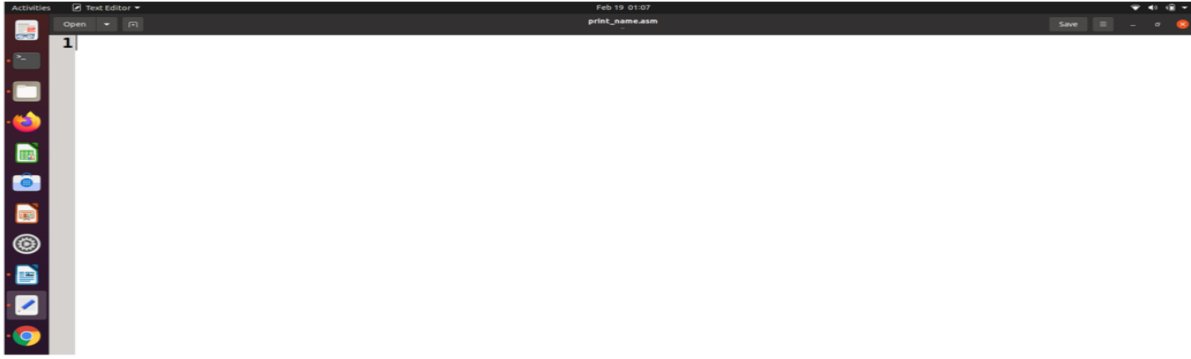
4.2.1 NASM सह पहिला असेम्बली प्रोग्राम

स्टेप 1: टर्मिनल उघडा आणि कमांड टाईप करा.

gedit print_name.asm

```
Terminal
$ gedit print_name.asm
```

येथे print _ name हे प्रोग्रामचे नाव आहे आणि. asm असेम्बली प्रोग्राम एक्सटेंशन साठी वापरला जातो.



या इंस्ट्रक्शन मुळे 'print_name.asm' या प्रोग्रामच्या नावाने 'gedit' विंडो उघडेल.

स्टेप 2: असेम्बली भाषेतील टिप्पण्या अर्धविरामाने सुरू होतात (;). प्रोग्रॅमची वाचनीयता आणि समज वाढवण्यासाठी प्रोग्रॅमचे तपशील आणि वाक्यरचना विस्तृत करण्यासाठी त्यांचा वापर केला जातो.

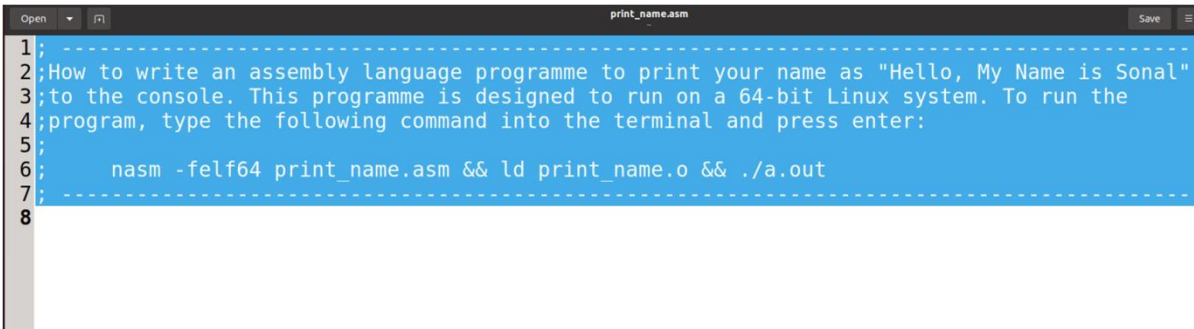
वाक्यरचनासह टिप्पण्या वापरून शिर्षकामध्ये कार्यक्रम कार्यान्वित करण्यासाठी काय करावे आणि आदेश द्यावा.? प्रोग्रामचे तपशील लिहा.

तथापि, हे एक ऐच्छिक स्टेप आहे. या अध्यायात, प्रोग्रॅमचे सर्व तपशील सुरुवातीला टिप्पण्या वापरून नमूद केले आहेत.

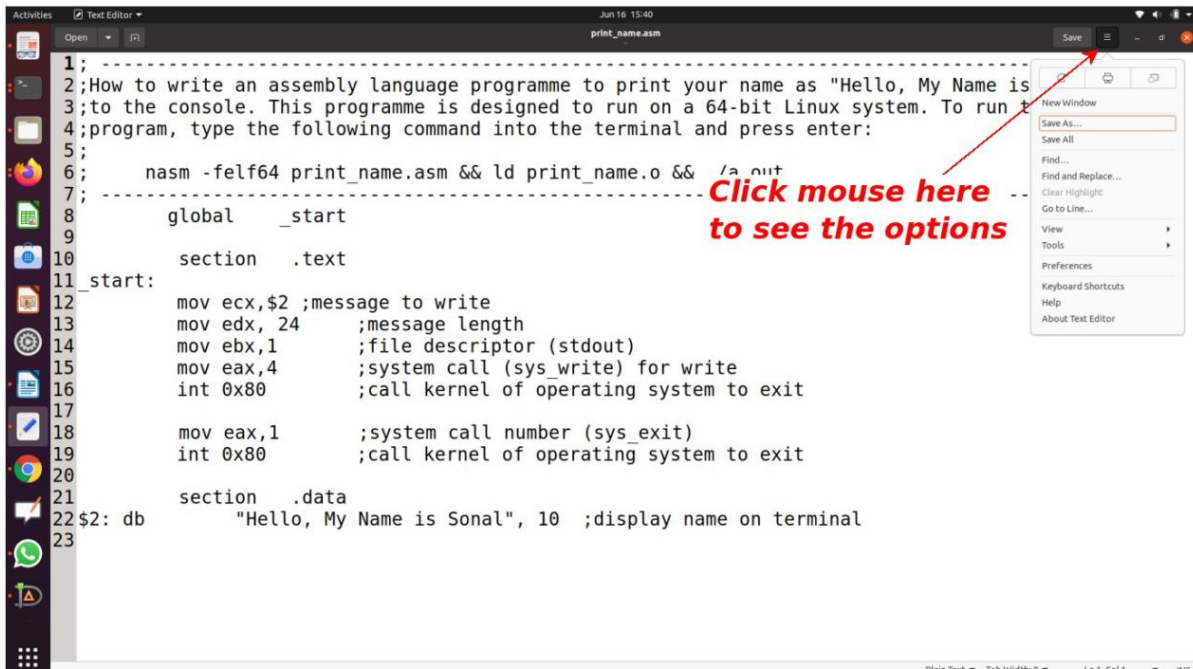


स्कॅन करा

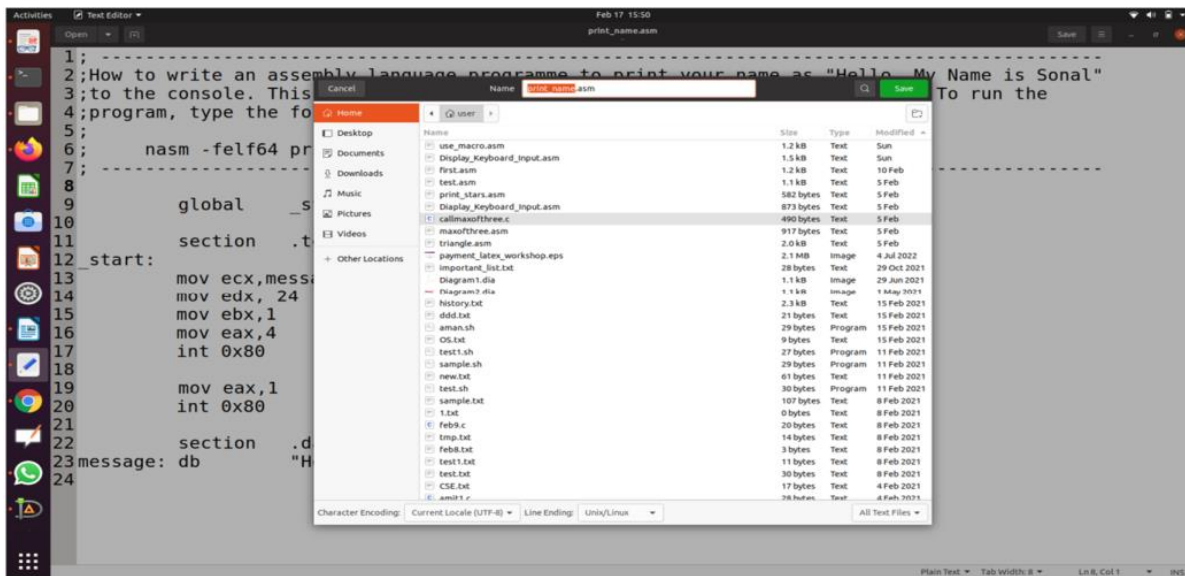
GUI द्वारे NASM
प्रतिष्ठापन शिकण्यासाठी



स्टेप 3: संपूर्ण असेम्बली लॉगवेज प्रोग्राम लिहा. सामान्य असेम्बली स्टेटमेंटमध्ये तीन फील्ड असतात: एक लेबल, ज्याला डारेक्टिवची ओळख म्हणून देखील ओळखले जाते, एक की, ज्याला असेम्बली इंस्ट्रक्शन किंवा असेंबलरला डारेक्टिव म्हणून देखील ओळखले जाते आणि एक टिप्पणी. ही फील्ड असेम्बली स्टेटमेंटची सामान्य रचना बनवतात. ही तीन फील्ड पूर्णपणे ऐच्छिक आहेत. तरीसुद्धा, प्रत्येक असेम्बली स्टेटमेंट वैध होण्यासाठी यापैकी किमान एक फील्ड आवश्यक आहे. पुढील स्टेप मध्ये हे तपशीलवार स्पष्ट केले आहे.



स्टेप 4: तुम्ही `ctrl+s` की वापरून प्रोग्राम सेव्ह करू शकता किंवा कीबोर्डमध्ये उपलब्ध असलेल्या `ctrl+शिफ्ट+s` कीज एकत्र वापरून वेगळ्या नावाने प्रोग्राम सेव्ह करू शकता. वैकल्पिकरित्या, तुम्ही माऊस कर्सरचा वापर करून वरील स्क्रीनशॉटच्या उजव्या बाजूला दर्शविल्याप्रमाणे तीन ओळींवर क्लिक करू शकता आणि तुम्हाला वेगवेगळे पर्याय लागू करण्याचे पर्याय दिसतील. म्हणून जतन करण्यासाठी, तुम्हाला खालील पर्याय मिळतील:



स्टेप 5: सेव्ह क्लिक केल्या नंतर, तुम्हाला प्रोग्रामचे नाव बदलण्याचा आणि तो कुठे सेव्ह केला जाईल ते स्थान प्रदर्शित करण्याचा पर्याय मिळेल. येथे, तुमच्याकडे सेव्हचे स्थान बदलण्याचा पर्याय देखील आहे.

स्टेप 6: लेबल हे असेम्बली स्टेटमेंटचे मजकूर ओळखकर्ता आहे. ब्रांच इंस्ट्रक्शन अंमलात आणताना ब्रांच उडीचे स्थान निर्दिष्ट करण्यासाठी लेबलचा वापर केला जातो. उदाहरणार्थ, एक लेबल खालीलप्रमाणे दर्शविले जाऊ शकते.

label: add r1, r2

लेबल आणि कोलन नंतर, असेम्बली इंस्ट्रक्शन "जोडा" लिहिली जाते आणि ऑपरेंडची यादी दिली जाते, i.e., r1, r2. लेबलमध्ये वैध अल्फा-न्यूमेरिक अक्षरे [a-z] [A-Z] [0-9] असू शकतात. मात्र, लेबलची सुरुवात अंकाने होऊ शकत नाही. पुढील स्क्रीनशॉटमध्ये print_name.asm प्रोग्राम लेबले ओळ क्रमांक 12 आणि 23 मध्ये ठळकपणे दर्शविली आहेत.

स्टेप 7: लिंकर global _start स्टेटमेंट शोधतो. मुळात, _start लिंकरला प्रोग्रामचा प्रवेश पॉइंट सांगतो.

स्टेप 8: असेंबलरचे डारेक्टिव एका कालावधीपासून सुरू होतात (.). याचा वापर नवीन विभाग सुरू करण्यासाठी किंवा स्थिरांक घोषित करण्यासाठी केला जातो. डारेक्टिव पॅरामीटर्सची यादी स्वीकारतो. नियमित संमेलनाच्या इंस्ट्रक्शन अक्षरांनी सुरू होतात.

```

1;-----
2;How to write an assembly language programme to print your name as "Hello, My Name is Sonal"
3;to the console. This programme is designed to run on a 64-bit Linux system. To run the
4;program, type the following command into the terminal and press enter:
5;
6;    nasm -felf64 print_name.asm && ld print_name.o && ./a.out
7;-----
8
9    global _start
10
11    section .text
12_start:
13    mov ecx,message
14    mov edx, 24
15    mov ebx,1    ;file descriptor (stdout)
16    mov eax,4    ;system call (sys_write) for write
17    int 0x80     ;call kernel of operating system to exit
18
19    mov eax,1    ;system call number (sys_exit)
20    int 0x80     ;call kernel of operating system to exit
21
22    section .data
23_message: db "Hello, My Name is Sonal", 10 ;display name on terminal
24

```

विभाग: असेम्बली प्रोग्राम खालील तीन वेगवेगळे विभाग वापरले जातात:

- ❖ **डेटा विभाग:** डेटा विभागात डेटा आणि स्थिरांक घोषित/आरंभीकृत केले जातात. हा डेटा रनटाइममध्ये बदलत नाही. हा विभाग म्हणून घोषित केला आहे.

section .data

- ❖ **बीएसएस विभाग:** व्यरिएबल BSS विभागात घोषित केले जातात. हा विभाग म्हणून घोषित केला आहे.

section .bss

- ❖ **टेक्स्ट विभाग:** प्रोग्रामची अंमलबजावणी येथे सुरू होते हे कर्नलला सांगण्यासाठी हा विभाग global _start च्या घोषणेसह सुरू होतो. खाली दर्शविल्याप्रमाणे लेबल _ स्टार्ट नंतर वास्तविक कोड लिहिला जातो:

section .text

global _start

_start:


```

1;
2;How to write an assembly language programme to print your name as "Hello, My Name is Sonal"
3;to the console. This programme is designed to run on a 64-bit Linux system. To run the
4;program, type the following command into the terminal and press enter:
5;
6;    nasm -felf64 print_name.asm && ld print_name.o && ./a.out
7;
8
9    global _start
10
11    section .text
12_start:
13    mov ecx,message
14    mov edx, 24
15    mov ebx,1
16    mov eax,4
17    int 0x80
18
19    section .data
20message: db "Hello, My Name is Sonal", 0
21
22
23
24

```

Must be declared for linker

Code segment of memory represented by .text section

Tell linker the entry point of the program

Data segment of memory represented by .data section

स्टेप 9: वापरकर्ता स्तरावरील प्रोग्राममधून सिस्टिम कॉलवर कंट्रोल ट्रान्सफर करण्यासाठी सिस्टिम कॉलसाठी विशेष इंस्ट्रक्शन वापरल्या जातात जसे की `int 0x80` ऑपरेटिंग सिस्टिम कर्नलला बाहेर पडण्यासाठी कॉल करण्यासाठी.

स्टेप 10: "EAX", "EBX", "ECX" आणि "EDX" नावाचे चार 32-बिट डेटा रजिस्टर एरिथमेटिक, लॉजिक आणि इतरांसारख्या विविध कार्यासाठी वापरले जातात. 'AX', 'BX', 'CX' आणि 'DX' हे 16-बिट डेटा रजिस्टर आहेत. "AH", "AL", "BH", "BL", "CH", "CL", "DH" आणि "DL" हे 8-बिट डेटा रजिस्टर म्हणून वापरले जाऊ शकतात. या रजिस्टर्समध्ये सिस्टिम कॉल आर्ग्युमेंट असतात.

```

1;
2;How to write an assembly language programme to print your name as "Hello, My Name is Sonal"
3;to the console. This programme is designed to run on a 64-bit Linux system. To run the
4;program, type the following command into the terminal and press enter:
5;
6;    nasm -felf64 print_name.asm && ld print_name.o && ./a.out
7;
8
9    global _start
10
11    section .text
12_start:
13    mov ecx,message
14    mov edx, 24
15    mov ebx,1
16    mov eax,4
17    int 0x80
18
19    mov eax,1
20    int 0x80
21
22    section .data
23message: db "Hello, My Name is Sonal", 10
24

```

Code snippet to use system call sys_write

Code snippet to use system call sys_exit

या रजिस्टरची वैशिष्ट्ये खालीलप्रमाणे आहेत:

- "AX". हा सेट्यक म्हणून वापरला जातो.
- अनुक्रमित ऍड्रेससाठी बेस रजिस्टर म्हणून "BX". चा वापर केला जातो.
- पुनरावृत्ती कार्यामध्ये काउन्टसाठी गणना रजिस्टर म्हणून "CX" चा वापर केला जातो.

• "DX". हा डेटा रजिस्टर म्हणून वापरला जातो.

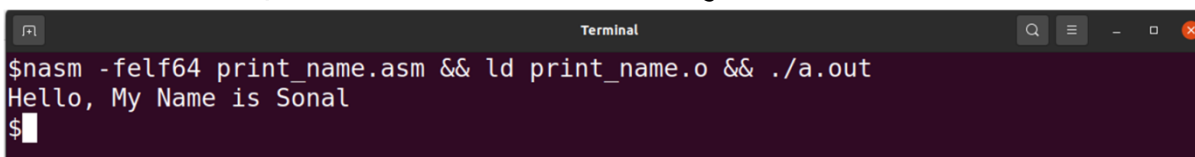
प्रत्येक syscall आणि त्याचा संबंधित क्रमांक (int 0x80 वर कॉल करण्यापूर्वी EAX मध्ये सेव्ह केलेला)/usr/include/asm-generic/unistd.h या फाईलमध्ये आढळू शकतो.

स्टेप 11: डी. बी. (डिफाइन बाइट) 1 बाइट वाटप करते यासारख्या स्टोरेज स्पेसमध्ये बाइट आरक्षित/प्रारंभ करण्यासाठी NASM डिफाइन असेंबलर डायरेक्टिव्ह वापरते.

स्टेप 12: खालील आदेश प्रोग्राम चालविण्यासाठी वापरला जातो. केवळ प्रोग्रामचे नाव (दिलेल्या उदाहरणात print_name.asm हे प्रोग्रामचे नाव आहे) बदलले जाईल, उर्वरित कमांड समान असेल.

nasm -felf64 print_name.asm && ld print_name.o && ./a.out

स्टेप 13: टर्मिनलवर print_name.asm प्रोग्रॅमचे आउटपुट दाखवा.



```
Terminal
$ nasm -felf64 print_name.asm && ld print_name.o && ./a.out
Hello, My Name is Sonal
$
```

4.3 असेम्बलर डारेक्टिव्ह

एसेम्बलर डारेक्टिव्चा वापर सोर्स प्रोग्रामला डेटा पुरवण्यासाठी केला जातो. उदाहरणार्थ, कॉन्स्टन्ट ची व्याख्या करण्यासाठी EQU डारेक्टिव्ह वापरला जातो.

CONSTANT_NAME EQU expression

समजा YEAR हे नाव 2023 चे व्हॅल्यू दर्शविण्यासाठी वापरले आहे.

YEAR EQU 2023

सोर्स प्रोग्रामचे ऑब्जेक्ट प्रोग्राममध्ये भाषांतर करताना, असेम्बलर YEAR हे नाव प्रोग्राममध्ये जिथे जिथे येते तिथे 2023 च्या व्हॅल्यूसह बदलतो. अशा विधानांना असेंबलर डायरेक्टिव्ह म्हणतात. खालील विधाने प्रोग्रॅममध्ये YEAR चा वापर स्पष्ट करतात:

mov ecx, YEAR

cmp eax, YEAR

प्रोग्राम 4.1:

use_directive.asm

EQU निर्देशाचा वापर स्पष्ट करण्यासाठी असेंबली लॉगवेज प्रोग्राम लिहा आणि टर्मिनलवर संदेश मुद्रित करा.

```

1;-----
2;How to write an assembly language programme to print the message
3;"Hello, programmers!
4;Start playing with Assembly. It is a language like all other languages!
5;Practice makes you perfect..."
6;using assembler directives to the console. This programme is designed to run on a 64-bit
7;Linux system. To run the program, type the following command into the terminal and
8;press enter:
9;
10;    nasm -felf64 use_directive.asm && ld use_directive.o && ./a.out
11;-----
12
13; Define assembler directives
14SYS_EXIT equ 1
15SYS_WRITE equ 4
16STDIN equ 0
17STDOUT equ 1
18
19; program code begins here
20    section .text
21    global _start
22
23_start:
24    mov eax, SYS_WRITE
25    mov ebx, STDOUT
26    mov ecx, str1
27    mov edx, len1
28    int 0x80
29
30    mov eax, SYS_WRITE
31    mov ebx, STDOUT
32    mov ecx, str2
33    mov edx, len2
34    int 0x80
35
36    mov eax, SYS_WRITE
37    mov ebx, STDOUT
38    mov ecx, str3
39    mov edx, len3
40    int 0x80
41
42    mov eax, SYS_EXIT
43    int 0x80 ;call kernel
44
45    section .data
46str1:    db 'Hello, programmers!',0xA,0xD
47len1:    equ $- str1
48str2:    db 'Start playing with Assembly. It is a language like all other languages!', 0xA,0xD
49len2:    equ $- str2
50str3:    db 'Practice makes you perfect...', 0xA,0xD
51len3:    equ $- str3

```

Output:

```

Terminal
$ nasm -felf64 use_directive.asm && ld use_directive.o && ./a.out
Hello, programmers!
Start playing with Assembly. It is a language like all other languages!
Practice makes you perfect...
$

```

4.4 प्रोसीजर आणि माक्रोस

एखादा प्रोग्राम इंस्ट्रक्शनच्या संचाची पुनरावृत्ती करतात. पुनरावृत्ती टाळण्यासाठी दोन पद्धती आहेत. एक म्हणजे विशिष्ट इंस्ट्रक्शनच्या संचासह एक प्रक्रिया विकसित करणे आणि जेव्हा आवश्यक असेल तेव्हा ती अमलात आणणे.

प्रोसीजर प्रोसीजरचा रिटर्न ऍड्रेस स्टॅक मध्ये सेटयित करतो. या प्रोसीजरचे आवाहन आणि रिटर्न ओव्हरहेड वेळेसाठी कारणीभूत ठरतात. तथापि, शिक्षणाच्या मोठ्या गटासाठी हे किमान आहे. इंस्ट्रक्शनच्या छोट्या गटांसाठी कार्यपद्धतींची शिफारस केली जात नाही कारण या प्रकरणात ओव्हरहेड आणि अंमलबजावणीची वेळ समतुल्य आहे. साध्या इंस्ट्रक्शनसाठी माक्रोला प्राधान्य दिले जाते. जेव्हा माक्रोचा वापर केला जातो, तेव्हा असेम्बलर इंस्ट्रक्शनसाठी मशीन कोड तयार करतो. माक्रोला कॉल करण्याची आणि परत करण्याची प्रक्रिया त्वरित असते. प्रत्येक माक्रो कॉल इनलाइन कोड तयार करतो, ज्यामुळे अतिरिक्त मेमोरीचा वापर होतो.

4.4.1 प्रक्रिया

जेव्हा डारेक्टिवचा विशिष्ट क्रम प्रोग्राममधील वेगवेगळ्या बिंदूंमध्ये पुनरावृत्ती केला जातो. प्रोग्रॅमतील इंस्ट्रक्शनचे हे अनुक्रम "उपप्रोग्राम" म्हणून लिहिले जाऊ शकतात ज्याला प्रोसीजर म्हणतात. "CALL" इंस्ट्रक्शन, मेमोरीतील प्रोसीजरच्या सुरुवातीच्या ऍड्रेससह, प्रत्येक वेळी इंस्ट्रक्शनच्या मालिकेची अंमलबजावणी करण्यासाठी वापरली जाऊ शकते. प्रोसीजरच्या शेवटी, एक "RET" इंस्ट्रक्शन असते, ज्यामुळे अंमलबजावणी मुख्य प्रोग्रॅमतील पुढील इंस्ट्रक्शनकडे जाते.

प्रक्रिया अशा प्रकारे "नेस्टेड" केल्या जाऊ शकतात की एक प्रक्रिया त्याच्या डारेक्टिव क्रमाचा भाग म्हणून दुसऱ्याला आवाहन करते. या प्रोसीजरचे फायदे आणि तोटे खालीलप्रमाणे आहेत:

फायदा:

1. प्रोसीजरतील इंस्ट्रक्शनच्या गटासाठी मशीन कोड केवळ एकदाच मेमोरीमध्ये लोड करणे आवश्यक आहे.

तोटा:

- 1) प्रोसीजर साठवणीसाठी स्टॅक आवश्यक आहे.
- 2) काही वेळेचे ओझे आहे. प्रक्रिया सुरू करण्यासाठी लागणारा वेळ आणि नंतर ती सुरू करणाऱ्या प्रोग्रॅमकडे परत जाणे.

प्रोग्राम 4.2:

procedure.asm

WCX आणि EDX रजिस्टरमध्ये साठवलेल्या संख्यांची वजाबाकी करण्यासाठी प्रक्रिया उप वापरून असेम्बली लॉगवेज प्रोग्राम लिहा. आउटपुट EAX रजिस्टरमध्ये साठवले जाते.

```

1; -----
2;How to write an assembly language programme to perform the subtraction of numbers
3;8 and 3 by calling a procedure sub. Display the result on the terminal.
4;This programme is designed to run on a 64-bit Linux system. To run the
5;program, type the following command into the terminal and press enter:
6;
7;    nasm -felf64 procedure.asm && ld procedure.o && ./a.out
8; -----
9
10     section .text
11     global _start
12 _start:
13     mov ecx, 8
14     sub ecx, 0
15     mov edx, 3
16     sub edx, 0
17     call sub        ;call sub procedure
18
19     mov [res], eax
20     mov ecx, msg
21     mov edx, len
22     mov ebx, 1      ;file descriptor (stdout)
23     mov eax, 4      ;system call number (sys_write)
24     int 0x80        ;call kernel
25
26     mov ecx, res
27     mov edx, 1
28     mov ebx, 1      ;file descriptor (stdout)
29     mov eax, 4      ;system call number (sys_write)
30     int 0x80        ;call kernel
31
32     mov ecx,newline ;message to write
33     mov edx,nlen    ;message length
34     mov ebx,1      ;file descriptor (stdout)
35     mov eax,4      ;system call number (sys_write)
36     int 0x80        ;call kernel
37
38
39     mov eax,1      ;system call number (sys_exit)
40     int 0x80        ;call kernel
41
42 sub:
43     mov eax, ecx
44     sub eax, edx
45     add eax, '0'
46     ret
47
48     section .data
49 msg: db "Subtraction using procedure, the output of 8-3 =", 0xA,0xD
50 len: equ $- msg
51 newline: db " ", 0xa    ; for the newline by the end of the program
52 nlen:    equ $ - newline ; length of message
53
54     segment .bss
55     res resb 1
56

```

Output:

```

Terminal
$ nasm -felf64 procedure.asm && ld procedure.o && ./a.out
Subtraction using procedure, the output of 8-3 =
5
$

```


4.4.2 माक्रो

जेव्हा इंस्ट्रक्शनचा पुनरावृत्ती सेट खूप लहान असतो, तेव्हा प्रोसीजरऐवजी 'माक्रो' वापरला जातो. माक्रो हा इंस्ट्रक्शनचा सेट आहे ज्याला प्रोग्रॅमच्या सुरुवातीला नाव दिले जाते. प्रत्येक वेळी 'माक्रो' चा वापर केला जातो तेव्हा 'कॉल' च्या ऐवजी असेंबलर इंस्ट्रक्शनचा निर्दिष्ट ब्लॉक समाविष्ट करतो. दुसऱ्या शब्दांत, माक्रो कॉल हे एक लघुरूप विधान आहे जे असेंबलरला सूचित करते, प्रत्येक वेळी जेव्हा प्रोग्राममध्ये माक्रो नाव दिसते, तेव्हा ते प्रोग्रामच्या सुरुवातीस त्या माक्रो म्हणून परिभाषित केलेल्या इंस्ट्रक्शनच्या गटाने बदलले जाते.

असेंबलर्स प्रत्येक माक्रो कॉलसाठी मशीन कोड तयार करतात. विस्तारामध्ये डारेक्टिव माक्रोची जागा घेतात. जनरेट केलेले मशीन कोड प्रोग्रामच्या अनुषंगाने असल्याने, असेंबलरला दूर जाऊन परत येण्याची गरज नाही. अशा प्रकारे, माक्रोचा वापर केल्याने प्रोसीजर कॉल आणि रिटर्न ओव्हरहेड दूर होते.

प्रत्येक वेळी माक्रोचा वापर केला जातो तेव्हा इन-लाइन कोड तयार करण्याचा तोटा असा आहे की एखादी प्रक्रिया वापरली जाते त्यापेक्षा प्रोग्राम अधिक मेमोरीचा वापर करेल. तक्ता 4.1 प्रोसीजर आणि माक्रो यांच्यातील तुलना दर्शवितो.

तक्ता 4.1 प्रक्रिया आणि माक्रो यांच्यातील तुलना

अनुक्रमांक	प्रोसीजर	माक्रो
1.	"कॉल" आणि "रिट" इंस्ट्रक्शनद्वारे प्रोग्राम अंमलबजावणी दरम्यान प्रवेश	परिभाषित माक्रोच्या नावाने असेंब्ली प्रोग्रॅमदरम्यान प्रवेशयोग्य.
2.	इंस्ट्रक्शनसाठीचा मशीन कोड केवळ एकदाच मेमोरीमध्ये साठवला जातो	जेव्हा माक्रोला कॉल केला जातो, तेव्हा इंस्ट्रक्शनसाठी मशीन कोड तयार केला जातो.
3.	प्रोसीजरसाठी कमी मेमोरीची आवश्यकता असते.	माक्रो वापरताना अधिक मेमोरी आवश्यक असते.
4.	पॅरामीटर्स रजिस्टर, मेमोरी लोकेशन्स किंवा स्टॅकमध्ये दिले जाऊ शकतात.	माक्रो विधानाचा भाग म्हणून प्रदान केलेले मापदंड.

कार्यान्वित करण्यासाठी CALL आणि RET इंस्ट्रक्शन नसल्यामुळे माक्रो अनुक्रम प्रोसीजरपेक्षा अधिक वेगाने कार्यान्वित होतात. माक्रो इंस्ट्रक्शन कोड कार्यान्वित झाल्यावर असेंबलरद्वारे कोडमध्ये समाविष्ट केल्या जातात. या प्रोसीजरला स्थूल विस्तार म्हणतात. माक्रो प्रोटोटाइप खालीलप्रमाणे लिहिला आहे.

```
%macro macro_name number_of_params
<macro body>
%endmacro
```

जेथे macro _ name हे macro चे नाव निर्दिष्ट करते, तेथे number _ of _ params हे पॅरामीटर्सची संख्या निर्दिष्ट करते.

प्रोग्राम 4.3:

use_macro.asm

```

1; -----
2;How to write an assembly language programme to print the message
3;"Hello, programmers!
4;Start playing with Assembly. It is a language like all other languages!
5;Practice makes you perfect..."
6;to the console. This programme is designed to run on a 64-bit Linux system. To run the
7;program, type the following command into the terminal and press enter:
8;
9;    nasm -felf64 use_macro.asm && ld use_macro.o && ./a.out
10; -----
11
12; A macro with two parameters, implements the write system call
13%macro display_message 2
14    mov eax, 4
15    mov ebx, 1
16    mov ecx, %1
17    mov edx, %2
18    int 80h
19%endmacro
20
21    section .text
22    global _start
23
24_start:
25    display_message str1, len1
26    display_message str2, len2
27    display_message str3, len3
28
29    mov eax,1 ;system call (sys_exit)
30    int 0x80 ;call kernel
31
32    section .data
33str1:    db 'Hello, programmers!',0xA,0xD
34len1:    equ $- str1
35str2:    db 'Start playing with Assembly. It is a language like all other languages!', 0xA,0xD
36len2:    equ $- str2
37str3:    db 'Practice makes you perfect...', 0xA,0xD
38len3:    equ $- str3

```

आउटपुट:

```

Terminal
$ nasm -felf64 use_macro.asm && ld use_macro.o && ./a.out
Hello, programmers!
Start playing with Assembly. It is a language like all other languages!
Practice makes you perfect...
$

```

4.5 असेम्बली प्रोग्राम

NASM मध्ये वापरल्या जाणाऱ्या विविध प्रकारच्या इंस्ट्रक्शन आणि त्यांची रचना दर्शविण्यासाठी खालील उपविभागांमध्ये सोप्या ते गुंतागुंतीच्या कार्यक्रमांचे स्पष्टीकरण दिले आहे.

4.5.1 सिम्पल प्रोग्राम:

साधे प्रोग्राम बेरीज, वजाबाकी करणे, गुणाकार इ. सारख्या एरिथमेटिक क्रियांचा वापर करतात. एरिथमेटिक इंस्ट्रक्शन आणि डेटा ट्रान्सफरची वाक्यरचना खालीलप्रमाणे आहे:



स्कॅन करा

NASM मधील साध्या प्रोग्राम साठी

प्रोग्राम 4.4:

display_keyboard_input.asm

```

1; -----
2;How to write an assembly language programme to read a number from the keyboard and display it
3;on the console screen. This programme is designed to run on a 64-bit Linux system. To run the
4;program, type the following command into the terminal and press enter:
5;
6;     nasm -felf64 display_keyboard_input.asm && ld display_keyboard_input.o && ./a.out
7; -----
8
9; Define assembler directives
10SYS_EXIT equ 1
11SYS_WRITE equ 4
12STDOUT equ 1
13
14
15%macro keyboard_input 2
16     mov eax, SYS_WRITE
17     mov ebx, STDOUT
18     mov ecx, %1
19     mov edx, %2
20     int 80h
21%endmacro
22
23     ;Code segment
24     section .text
25     global _start
26
27 _start:
28
29     keyboard_input userMsg, lenUserMsg
30
31     ;Read and store the user input
32     mov eax, 3
33     mov ebx, 2
34     mov ecx, num
35     mov edx, 8 ; up to 8 bytes of data can store
36     int 80h
37
38     ;Output the message 'The entered number is: '
39     keyboard_input dispMsg, lenDispMsg
40
41     ;Output the number entered
42     keyboard_input num, 8
43
44     mov eax, SYS_EXIT ; Exit
45     mov ebx, 0
46     int 80h
47
48     ;Data segment
49     section .data
50 userMsg: db 'Please enter a number: ' ;Ask the user to enter a number
51 lenUserMsg: equ $-userMsg ;The length of the message
52 dispMsg: db 'You have entered: '
53 lenDispMsg: equ $-dispMsg
54
55     section .bss ;Uninitialized data
56     num resb 8

```

आउटपुट:

```

Terminal
$ nasm -felf64 display_keyboard_input.asm && ld display_keyboard_input.o && ./a.out
Please enter a number: 1234567
You have entered: 1234567
$

```


4.5.2 अरीथमाटिक प्रोग्राम्स:

अरीथमाटिक इंस्ट्रक्शन बेरीज, वजाबाकी, गुणाकार आणि भागाकार कार्ये करतात. एरिथमेटिक इंस्ट्रक्शन आणि डेटा ट्रान्स्फरची वाक्यरचना खालीलप्रमाणे आहे.



स्कॅन करा
गुणाकार आणि विभाजन
प्रोग्राम साठी

तक्ता 4.2: एरिथमेटिक क्रियांची यादी

ऑपेरेशन	ऑपेरंड	टिप्पण्या
ADD/SUB	डिस्टिनेशन, सोर्स	डिस्टिनेशन, सोर्स रजिस्टरमध्ये साठवलेल्या संख्यांची बेरीज/वजाबाकी करा. डिस्टिनेशन रजिस्टरमध्ये निकाल साठवले जातात.
MUL/IMUL	गुणक	साइन / अनसाइन गुणाकार करा
DIV/IDIV	विभाजक	साइन / अनसाइन विभाजक करा
INC/DEC	डिस्टिनेशन	ऑपरेंडला एकाने वाढवणे/कमी करणे
Mov	डिस्टिनेशन, सोर्स	डेटा सोर्स डिस्टिनेशन कडे ट्रान्स्फर करणे

प्रोग्राम 4.5:

add_sub.asm

```

1; -----
2;How to write an assembly language programme to add and sub the values 5 and 3.
3;What is the value in AL register after performing addition and subtraction operation.
4;This programme is designed to run on a 64-bit Linux system. To run the
5;program, type the following command into the terminal and press enter:
6;
7;    nasm -felf64 add_sub.asm && ld add_sub.o && ./a.out
8; -----
9
10     section .text
11     global _start
12 _start:    ;tell linker entry point
13
14     mov ecx,message ;message to write
15     mov edx,len     ;message length
16     mov ebx,1       ;file descriptor (stdout)
17     mov eax,4       ;system call number (sys_write)
18     int 0x80        ;call kernel
19
20     mov al, 5        ;store 5 in the al
21     mov bl, 3        ;store 3 in the bl
22     add al, bl       ;add al and bl registers, the output is stored in al register
23     add al, byte '0';converting decimal to ascii
24     mov [result], al
25
26     mov eax, 4
27     mov ebx, 1
28     mov ecx, result
29     int 0x80        ;call kernel
30
31     mov ecx,newline ;message to write
32     mov edx,nlen     ;message length
33     mov ebx,1       ;file descriptor (stdout)
34     mov eax,4       ;system call number (sys_write)
35     int 0x80        ;call kernel
36
37     mov ecx,message1 ;message to write
38     mov edx,len1     ;message length
39     mov ebx,1       ;file descriptor (stdout)
40     mov eax,4       ;system call number (sys_write)
41     int 0x80        ;call kernel
42
43     mov al, 5        ;store 5 in the al
44     mov bl, 3        ;store 3 in the bl
45     sub al, bl       ;sub al and bl registers, the output is stored in al register
46     add al, byte '0';converting decimal to ascii
47     mov [result], al
48
49     mov eax, 4
50     mov ebx, 1
51     mov ecx, result
52     int 0x80        ;call kernel
53
54
55     mov ecx,newline ;message to write
56     mov edx,nlen     ;message length
57     mov ebx,1       ;file descriptor (stdout)
58     mov eax,4       ;system call number (sys_write)
59     int 0x80        ;call kernel
60
61     mov eax,1        ;system call number (sys_exit)
62     int 0x80        ;call kernel
63
64
65     section .data
66 message: db "Addition of (5+3):", 0xa      ; display the message on the terminal
67 len:     equ $ - message                  ; length of message
68
69 message1: db "Subtraction of (5-3):", 0xa  ; display the message on the terminal
70 len1:     equ $ - message1                ; length of message
71
72 newline: db "\n", 0xa                    ; for the newline by the end of the program
73 nlen:     equ $ - newline                 ; length of message
74
75     section .bss
76 result: resb 1
77

```

आउटपुट:

```

Terminal
$ nasm -felf64 add_sub.asm && ld add_sub.o && ./a.out
Addition of (5+3):
8
Subtraction of (5-3):
2
$

```

4.5.3 लॉजिकल इंस्ट्रक्शन

लॉजिक इंस्ट्रक्शन डेटा प्रोसेस करण्या करीता वापरले जातात उदाहरणार्थ बिटवाईस or, आणि, एक्सक्लुझिव्ह or , आणि not इंस्ट्रक्शन.

तक्ता 4.3: लॉजिकल ऑपरेशन ची यादी

ऑपरेशन	ऑपरेंड	टिप्पण्या
AND/OR/XOR	ऑपरेंड1, ऑपरेंड 2	ऑपरेंड 1 आणि ऑपरेंड 2 दरम्यान लॉजिकल AND/OR/XOR ऑपरेशन करा. परिणाम ऑपरेंड 1 मध्ये संग्रहित केला आहे
NOT	ऑपरेंड1	लॉजिकल नॉट ऑपरेशन करा आणि परिणाम ऑपरेंड 1 मध्ये सेटयित करा

प्रोग्राम 4.6:

[logical_xor_operation.asm](#)

```

1;-----
2;How to write an assembly language programme to store the value 5 and 3 in the AL and the BL
3;register respectively. What is the value in AL register after performing XOR AL, BL operation.
4;This programme is designed to run on a 64-bit Linux system. To run the
5;program, type the following command into the terminal and press enter:
6;
7;    nasm -felf64 logical_xor_operation.asm && ld logical_xor_operation.o && ./a.out
8;-----
9
10    section .text
11    global _start    ;must be declared for using gcc
12_start:             ;tell linker entry point
13
14    mov eax,4         ;system call number (sys_write)
15    mov ebx,1         ;file descriptor (stdout)
16    mov ecx,message   ;message to write
17    mov edx,len       ;message length
18    int 0x80          ;call kernel
19
20    mov al, 5         ;store 5 in the al
21    mov bl, 3         ;store 3 in the bl
22    xor al, bl        ;xor al and bl registers, the output is stored in al register
23    add al, byte '0';converting decimal to ascii
24    mov [result], al
25
26    mov eax, 4
27    mov ebx, 1
28    mov ecx, result
29    int 0x80          ;call kernel
30
31
32    mov eax,4         ;system call number (sys_write)
33    mov ebx,1         ;file descriptor (stdout)
34    mov ecx,newline   ;message to write
35    mov edx,nlen      ;message length
36    int 0x80          ;call kernel
37
38
39    mov eax,1         ;system call number (sys_exit)
40    int 0x80          ;call kernel
41
42    section .data
43message: db "Output of (5 XOR 3):", 0xa    ; display the message on the terminal
44len:     equ $ - message                  ; length of message
45
46newline: db " ", 0xa                    ; for the newline by the end of the program
47nlen:    equ $ - newline                  ; length of message
48
49    section .bss
50result: resb 1
51
52
53

```

आउटपुट:

```

Terminal
$ nasm -felf64 logical_xor_operation.asm && ld logical_xor_operation.o && ./a.out
Output of (5 XOR 3):
6
$

```

4.5.4 ब्रँच इंस्ट्रक्शन

ब्रँच इंस्ट्रक्शन प्रोग्रॅमच्या अंमलबजावणीवरील प्रोग्रॅम चा कंट्रोल वेगवेगळ्या भागांवर ढकलतात. 'फॉर' लूप्स आणि 'इफ-देन-एल्स' विधाने ही ब्रान्च इंस्ट्रक्शनची उदाहरणे आहेत. ब्रॅन्च इंस्ट्रक्शन अन-कंडिशनल किंवा कंडिशनल जम्प असू शकतात.

1. अन कंडिशनल जम्प करताना, 'JMP' इंस्ट्रक्शन, डारेक्टिवच्या सध्याच्या अंमलबजावणीला वगळून, प्रोग्रॅमच्या अंमलबजावणीचे कंट्रोल डारेक्टिवच्या



स्कॅन करा
ब्रँच इंस्ट्रक्शनच्या
प्रोग्रॅम साठी

वेगळ्या पॉइंटवर ट्रान्सफर करते.

- कंडिशनल जम्प मध्ये, "CMP" इंस्ट्रक्शन स्थिती तपासते आणि स्थितीवर आधारित प्रोग्रॅमतील जम्प इंस्ट्रक्शनच्या उपलब्ध श्रेणीतून योग्य जम्प इंस्ट्रक्शन वापरते. कंडिशनल जम्प अनुक्रमिक अंमलबजावणीच्या प्रवाहात व्यत्यय आणते आणि कंट्रोल नवीन ठिकाणी ट्रान्सफर करते.

"CMP" इंस्ट्रक्शन कंडिशनल जम्प इंस्ट्रक्शनच्या संयोगाने वापरली जाते. ऑपरेंड समान आहेत की नाही हे निर्धारित करण्यासाठी ते एका ऑपरेंडला दुसऱ्यापासून वजा करते.

CMP destination, source

डेस्टिनेशन ऑपरेंड एकतर रजिस्टर किंवा मेमोरी असू शकतो, तर सोर्स ऑपरेंड एकतर स्थिर (त्वरित) डेटा, रजिस्टर किंवा मेमोरी असू शकतो. उदाहरणादाखल,

```
CMP DX, 00    ; Compare the DX value with zero
JE L7         ; If yes, then jump to label L7
.
.
L7: ...
```

प्रोग्राम 4.7:array_sum.asm

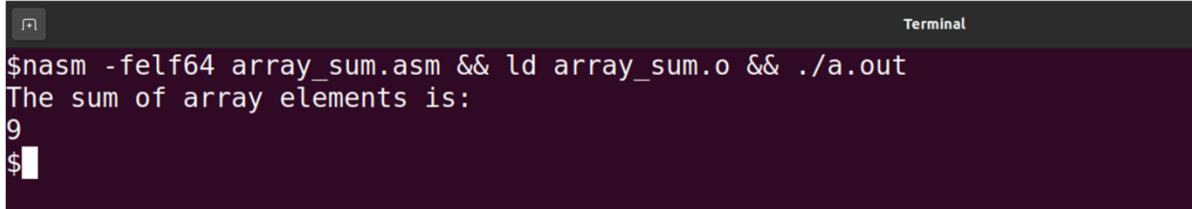
```
1);-----
2;Write an assembly language program to store four values in array x: 1, 4, 3, and 1.
3;Perform the addition of the array elements and display the sum.
4;This programme is designed to run on a 64-bit Linux system. To run the
5;program, type the following command into the terminal and press enter:
6;
7;    nasm -felf64 array_sum.asm && ld array_sum.o && ./a.out
8;-----
9
10; Define assembler directives
11SYS_EXIT equ 1
12SYS_WRITE equ 4
13STDOUT equ 1
14
15
16%macro display_message 2
17    mov eax, SYS_WRITE
18    mov ebx, STDOUT
19    mov ecx, %1
20    mov edx, %2
21    int 80h
22%endmacro
23
24
25    section .text
26    global _start    ;must be declared for linker (ld)
27_start:
28
```

```

29      mov eax,4          ;number bytes to be summed
30      mov ebx,0          ;EBX will store the sum
31      mov ecx, x          ;ECX will point to the current element to be summed
32
33top:   add ebx, [ecx]
34      add ecx,1          ;move pointer to next element
35      dec eax            ;decrement counter
36      jnz top            ;if counter not 0, then loop again
37
38done:  add ebx, '0'
39      mov [sum], ebx ;done, store result in "sum"
40
41      display_message msg, len
42
43      display_message sum, 1
44
45      display_message newline, nlen
46
47      mov eax, STDOUT ;system call number (sys_exit)
48      int 0x80 ;call kernel
49
50
51      section .data
52msg db "The sum of array elements is: ", 0xA,0xD
53len equ $- msg
54
55newline: db " ", 0xa      ; for the newline by the end of the program
56nlen:    equ $ - newline  ; length of message
57
58      global x
59x:
60db 1
61db 4
62db 3
63db 1
64sum:
65db 0
66

```

आउटपुट:



```

Terminal
$ nasm -felf64 array_sum.asm && ld array_sum.o && ./a.out
The sum of array elements is:
9
$

```

4.5.5 एरिथमेटिक एक्सप्रेसन इन्हालुशन

इन्फिक्स नोटेशनचा वापर एक्सप्रेसनचे वर्णन करण्यासाठी केला जातो, ज्यामध्ये बेरीज, वजाबाकी, गुणाकार, भागाकार आणि यासारखे एरिथमेटिक ऑपरेटर दोन ऑपरेंडमध्ये लिहिले जातात, i.e., "A + B". "A + (B * C)" पासून "(A + B) * C" वेगळे करण्यासाठी, या संकेतनाला कंस किंवा ऑपरेटर प्राधान्य आवश्यक आहे.

प्रोग्राम 4.8:

evaluate_expression.asm


```

1; -----
2;How to write an assembly language programme to evaluate the expression 8/4+2x2-1
3;What is the value in AL register after evaluating given expression.
4;This programme is designed to run on a 64-bit Linux system. To run the
5;program, type the following command into the terminal and press enter:
6;
7;    nasm -felf64 evaluate_expression.asm && ld evaluate_expression.o && ./a.out
8; -----
9
10     section .text
11     global _start
12 _start:                ;tell linker entry point
13
14     mov ecx,message ;message to write
15     mov edx,len      ;message length
16     mov ebx,1        ;file descriptor (stdout)
17     mov eax,4        ;system call number (sys_write)
18     int 0x80         ;call kernel
19
20     mov al, 8         ;store 8 in the al
21     mov bl, 4         ;store 4 in the bl
22     div bl            ;divide dividend stored in al by divisor stored in bl register
23     add al, 2         ;add intermediate result stored in al with value 2
24     mov cl, 2
25     mul cl            ;divide multiplicand stored in al by multiplier stored in bl register
26     sub al, 1         ;subtract intermediate result stored in al with value 1
27     add al, byte '0' ;converting decimal to ascii
28     mov [result], al
29
30     mov eax, 4
31     mov ebx, 1
32     mov ecx, result
33     int 0x80         ;call kernel
34
35     mov ecx,newline ;message to write
36     mov edx,nlen     ;message length
37     mov ebx,1        ;file descriptor (stdout)
38     mov eax,4        ;system call number (sys_write)
39     int 0x80         ;call kernel
40
41
42     mov eax,1        ;system call number (sys_exit)
43     int 0x80         ;call kernel
44
45     section .data
46 message: db "The output of arithmetic expression (8/4+2x2-1) is :", 0xa ;display message
47 len:     equ $ - message ; length of message
48
49 newline: db " ", 0xa ; for the newline by the end of the program
50 nlen:     equ $ - newline ; length of message
51
52     section .bss
53 result: resb 1 ;reserve 1 byte
54

```

आउटपुट:

```

Terminal
$ nasm -felf64 evaluate_expression.asm && ld evaluate_expression.o && ./a.out
The output of arithmetic expression (8/4+2x2-1) is :
7
$

```

4.5.6 स्ट्रिंग मानिपुलेशन

स्ट्रिंग हाताळणीच्या कृतींमध्ये स्ट्रिंगची कॉपी करणे, स्ट्रिंग उलट करणे, अक्षर मोजणे इ. समाविष्ट आहे [1]. 32-बिट माहितीसाठी स्ट्रिंग इंस्ट्रक्शन अनुक्रमे सोर्स आणि डेस्टिनेशन ऑपरेंडचा संदर्भ घेण्यासाठी ESI आणि EDI

रजिस्टरचा वापर करतात. तथापि, 16-बिट डेटासाठी, SI आणि DI रजिस्टर अनुक्रमे सोर्स आणि डेस्टिनेशन दर्शविण्यासाठी वापरले जातात. स्ट्रिंग प्रोसेसिंगमध्ये पाच मूलभूत इंस्ट्रक्शन असतात.

- MOVS इंस्ट्रक्शन मेमोरी ऍड्रेसदरम्यान 1 बाइट, वर्ड किंवा डबल-वर्ड डेटा ट्रान्सफर करते.
- LOAD इंस्ट्रक्शन मेमोरी-आधारित डेटा पुनर्प्राप्ति करते. AL रजिस्टर एकाच बाइट ऑपरेंडसह लोड केले जाते, AX रजिस्टर एकाच वर्ड ऑपरेंडसह लोड केले जाते आणि EAX रजिस्टर दुहेरी वर्ड ऑपरेंडसह लोड केले जाते.
- STOS इंस्ट्रक्शन रजिस्टर (AL, AX, किंवा EAX) पासून मेमोरीमध्ये डेटा लिहितो.
- CMPS इंस्ट्रक्शन मेमोरी-आधारित डेटाच्या दोन घटकांची तुलना करते. डेटाच्या आकारांमध्ये बाइट, वर्ड आणि दुहेरी वर्ड यांचा समावेश होतो.
- SCAS इंस्ट्रक्शन मेमोरीच्या स्थानाच्या विरुद्ध रजिस्टरच्या (AL, AX किंवा EAX) सामग्रीचे व्हॅल्यूकन करते.

प्रोग्राम 4.9:

```

1; -----
2;In cryptography, a caesar cipher encrypts a data by simply replacing each alphabet in it
3;with a shift of two alphabets, so a will be substituted by c, b with d and so on.
4;Write an assembly language program to encrypt the string 'assembler'.
5;This programme is designed to run on a 64-bit Linux system. To run the
6;program, type the following command into the terminal and press enter:
7;
8;     nasm -felf64 encrypt.asm && ld encrypt.o && ./a.out
9; -----
10
11
12section .text
13    global _start                ;must be declared for using gcc
14
15_start:                          ;tell linker entry point
16    mov ecx, len
17    mov esi, s1
18    mov edi, s2
19
20loop_continue:
21    lodsb                        ; loads byte into the al register from memory
22    add al, 02
23    stosb                        ; stores byte from register to memory
24    loop    loop_continue
25    cld                          ; clear direction flag (DF)

```

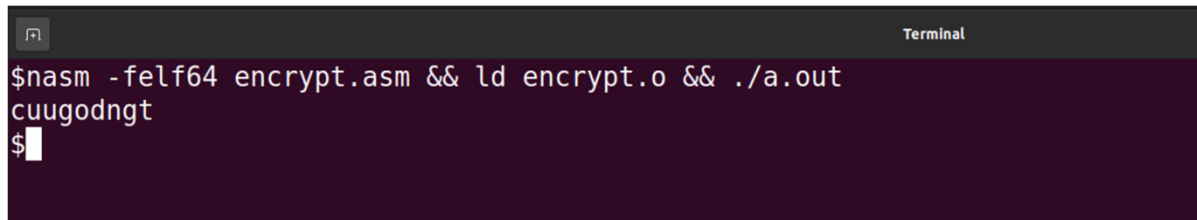


```

26 rep      movsb      ; moves 1 byte
27
28 mov edx,20 ;message length
29 mov ecx,s2 ;message to write
30 mov ebx,1  ;file descriptor (stdout)
31 mov eax,4  ;system call number (sys_write)
32 int 0x80   ;call kernel
33
34 mov ecx,newline ;message to write
35 mov edx,nlen ;message length
36 mov ebx,1  ;file descriptor (stdout)
37 mov eax,4  ;system call number (sys_write)
38 int 0x80   ;call kernel
39
40 mov eax,1  ;system call number (sys_exit)
41 int 0x80   ;call kernel
42
43 section .data
44 s1 db 'assembler', 0 ;source
45 len equ $-s1
46
47 newline: db " ", 0xa ; for the newline by the end of the program
48 nlen: equ $ - newline ; length of message
49
50 section .bss
51 s2 resb 10 ;reserve 10 bytes for destination
52

```

आउटपुट:



```

Terminal
$ nasm -felf64 encrypt.asm && ld encrypt.o && ./a.out
cuugodngt
$

```

4.5.7 सॉर्टिंग

सॉर्टिंग करणे ही संगणक विज्ञानातील एक महत्वाची आणि अनेकदा वापरली जाणारी प्रोसीजर आहे. NASM असेंबलरमध्ये सॉर्टिंग हे बबल सॉर्ट, इन्सर्शन सॉर्ट, सिलेक्शन सॉर्ट आणि क्विकसॉर्ट यासह अनेक अल्गोरिदमच्या वापराद्वारे पूर्ण केले जाऊ शकते. बबल सॉर्टची प्रक्रिया प्रोग्राम 4.10 मध्ये दर्शविली आहे.

बबल सॉर्ट शेजारच्या घटकांची तुलना करते आणि ते खराब असल्यास त्यांची अदलाबदल करते. यादीची क्रमवारी होईपर्यंत ती पुन्हा पुन्हा फिरवली जाते.

प्रोग्राम 4.9:

sorting_integers.asm

```

1; -----
2;Write an assembly language programme to sort unsorted items 9 8 7 6 5 4 3 2 1 0.
3;The sorted elements list 0 1 2 3 4 5 6 7 8 9 should be displayed on the terminal.
4;This programme is designed to run on a 64-bit Linux system. To run the program,
5;type the following command into the terminal and press enter:
6;
7;    nasm -felf64 sorting_integers.asm && ld sorting_integers.o && ./a.out
8; -----
9
10section .text
11
12startnew:
13
14    mov ecx,newline
15    mov edx,1
16    mov eax,4
17    mov ebx,1
18    int 0x80
19    ret
20
21convert:
22    mov ecx,[inp]
23    mov bl,10
24    mov al,cl
25    sub al,30h
26
27    cmp ch,10
28    je converted
29    mul bl
30    sub ch,30h
31    add al,ch
32
33converted:
34    mov byte[inp],al
35    ret
36
37takeinput:
38    mov ecx,msg_element
39    mov edx,length_msg_element
40    mov eax,4
41    mov ebx,1
42    int 0x80
43
44terminal_input:
45
46    mov ecx,inp
47    mov edx,3
48    mov eax,3
49    mov ebx,0
50    int 0x80
51
52    call convert
53
54    ret
55

```

```

56 global _start
57 _start:
58
59     mov ecx,num_elements
60     mov edx,length_num_elements
61     mov eax,4
62     mov ebx,1
63     int 0x80
64
65     call terminal_input
66
67     mov al,byte[inp]
68     mov byte[n],al
69
70     mov ebx,0
71     mov bl,[n]
72     mov edx,0
73 taking:
74     push dx
75     call takeinput
76     mov al,byte[inp]
77     mov edx,0
78     pop dx
79     mov byte[input+edx],al
80     add dx,1
81     mov bl,[n]
82     cmp dl,bl
83     jne taking
84
85     mov edx,0
86 sort:
87     mov bl,byte[input+eax]
88     mov cl,byte[input+eax+1]
89     cmp bl,cl
90     jl no_swap
91     mov byte[input+eax+1],bl
92     mov byte[input+eax],cl
93 no_swap:
94     add eax,1
95     mov bl,byte[n]
96     sub bl,1
97     cmp al,bl
98     jne sort
99     mov eax,0
100    add edx,1
101    mov bl,byte[n]
102    add bl,1
103    cmp dl,bl
104    jne sort
105
106    call startnew
107
108    mov ecx,sorted_list_msg

```

```

109     mov edx,length_sorted_list_msg
110     mov eax,4
111     mov ebx,1
112     int 0x80
113
114     mov edx,0
115     mov esi,0
116
117 output:
118
119     push 29h
120     mov ebx,0
121     mov bl,byte[input+esi]
122     mov ax,bx
123     mov ebx,0
124     mov bl,10
125 break:
126     mov edx,0
127     div bx
128     add dl,30h
129     push dx
130     cmp al,0
131     jne break
132
133     mov edx,0
134     pop dx
135
136 prints:
137
138     mov byte[printdata],dl
139
140     mov ecx,printdata
141     mov edx,1
142     mov eax,4
143     mov ebx,1
144     int 0x80
145
146     mov edx,0
147     pop dx
148     cmp dl,29h
149     jne prints
150
151     mov ecx,space
152     mov edx,1
153     mov eax,4
154     mov ebx,1
155     int 0x80
156
157
158     add esi,1
159     mov ebx,0
160     mov bl,byte[n]
161     cmp esi,ebx
162     jne output
163
164     call startnew
165     call startnew
166
167     mov eax,1
168     mov ebx,0
169     int 0x80
170
171 section .bss
172     input resb 100
173
174 section .data
175     inp dd 30h
176     temp db 30h
177     space db 32
178     newline db 10
179     n db 30h
180     printdata db 30h
181     num_elements db "Enter the number of elements: ",32
182     length_num_elements equ $-num_elements
183     msg_element db "Enter the data: ",32
184     length_msg_element equ $-msg_element
185     sorted_list_msg db "Sorted elements list",10
186     length_sorted_list_msg equ $-sorted_list_msg

```

आउटपुट:

A terminal window titled "Terminal" with a dark background and light text. It shows the execution of a program. The first line is a command: "\$nasm -felf64 sorting_integers.asm && ld sorting_integers.o && ./a.out". The next line is a prompt "Enter the number of elements: 10". This is followed by ten lines of prompts "Enter the data:" with values 9, 8, 7, 6, 5, 4, 3, 2, 1, and 0. Then, it shows "Sorted elements list" and the output "0 1 2 3 4 5 6 7 8 9". The prompt "\$" is visible at the bottom left.

```
$nasm -felf64 sorting_integers.asm && ld sorting_integers.o && ./a.out
Enter the number of elements: 10
Enter the data: 9
Enter the data: 8
Enter the data: 7
Enter the data: 6
Enter the data: 5
Enter the data: 4
Enter the data: 3
Enter the data: 2
Enter the data: 1
Enter the data: 0

Sorted elements list
0 1 2 3 4 5 6 7 8 9

$
```

युनिट सारांश:

- मशीन इंस्ट्रक्शन एसेम्बलरमध्ये कोडित केल्या जाऊ शकतात. सर्वसाधारणपणे, असेम्बली लॉगवेज प्रोग्रामिंगमधील एक विधान हे एका मशीन इंस्ट्रक्शनच्या बरोबरीचे असते. असेम्बलर असेम्बली भाषेत लिहिलेल्या प्रोग्राम्सचे मशीन कोडमध्ये रूपांतर करतो.
- असेम्बली लॉगवेज ISA आणि असेम्बली -विशिष्ट आहेत.
- ऑपरेटिंग सिस्टमला समर्थन देणाऱ्या कर्नल आणि डिव्हाइस ड्रायव्हर्ससाठी प्रोग्राम डिझाइन करण्यासाठी असेम्बली लॉगवेज वापरली जाते.
- असेम्बली लॉगवेज हार्डवेअर डिझायनर्सना ISA सीम्यांटिक समजून घेण्यास मदत करतात. हे रचनेसाठी दिशा प्रदान करते.
- NASM असेम्बलर लिनक्स आणि विंडोज या दोन्ही ऑपरेटिंग सिस्टमवर डाउनलोड करण्यासाठी विनामूल्य आहे.
- NASM हा प्रोग्राम "Nasm-felf64 filename.asm & & ld filename.o & ./a.out" ह्या कमांडचा वापर करून चालवतो. असेम्बली प्रोग्रामचा डेटा विभाग डेटा प्रारंभ करतो, बीएसएस विभाग व्हेरिएबल्स परिभाषित करतो आणि मजकूर विभागात वास्तविक कोड असतो.
- एसेम्बलर डारेक्टिव प्रोग्रामरला सोर्स प्रोग्रामचे ऑब्जेक्ट प्रोग्राममध्ये भाषांतर करण्यासाठी आवश्यक असलेली इतर माहिती निर्दिष्ट करण्याची परवानगी देतात. स्थिरांकांची व्याख्या करण्यासाठी EQU डारेक्टिव वापरला जातो.
- जेव्हा प्रोग्रॅममध्ये पुनरावृत्ती केलेल्या इंस्ट्रक्शनचा समूह आणि या इंस्ट्रक्शनचा साईझ खूप मोठा असतो तेव्हा प्रक्रिया असेम्बली भाषेत परिभाषित केली जाते.
- जेव्हा इंस्ट्रक्शनचा पुनरावृत्ती गट खूप लहान असतो किंवा प्रक्रिया म्हणून लिहिणे योग्य नसते, तेव्हा 'माक्रो' वापरला जातो.
- प्रोग्रॅमच्या सुरुवातीला माक्रो म्हणून परिभाषित केलेल्या इंस्ट्रक्शनचा गट. प्रोग्रॅमतील प्रत्येक स्थूल नावासाठी, इंस्ट्रक्शनच्या गटासाठी मशीन कोडसह ते बदला.
- साधे असेम्बली प्रोग्राम जोड, वजाबाकी, विभागणी, गुणाकार यासारखी सोपी एरिथमेटिक कार्ये करतात. कोणत्याही एरिथमेटिक अभिव्यक्तीमध्ये एकापेक्षा जास्त एरिथमेटिक क्रिया असतात. तर हे गुंतागुंतीचे असेम्बली प्रोग्राम आहेत.
- ब्रांच इंस्ट्रक्शन प्रोग्रॅमच्या अंमलबजावणीवरील कंट्रोल प्रोग्रॅमच्या वेगवेगळ्या भागांवर ढकलतात. 'FOR' LOOP आणि 'IF-THEN-ELSE' विधाने ही ब्रांच इंस्ट्रक्शनची उदाहरणे आहेत.
- असेम्बली भाषेचे प्रोग्राम स्ट्रिंग हाताळणीसाठी लिहिले जातात. जसे की स्ट्रिंग कॉपी करणे, स्ट्रिंग उलट करणे, अक्षरे मोजणे इ.

स्वाध्याय:**बहुपर्यायी प्रश्न**

प्रश्न 4.1 असेम्बलर _____ मध्ये लिहिलेल्या प्रोग्रामला मशीन इंस्ट्रक्शनमध्ये रूपांतरित करते.

अ) C लॉगवेज (ब) C++ लॉगवेज (क) असेम्बली लॉगवेज (ड) पायथन लॉगवेज

प्रश्न 4.1 _____ इंस्ट्रक्शन दोन संख्यांची भर घालते.

अ) MOV (ब) ADD (क) SUB (ड) DIV

प्रश्न 4.3 असेम्बलर डारेक्टिव Sum EQU 200 _____ करते

(अ) रक्कमेच्या पहिल्या घटनेला 200 व्हॅल्यू नियुक्त करते.

(ब) बेरीजच्या प्रत्येक घटनेला 200 व्हॅल्यू नियुक्त करते.

(क) त्याच्या मूळ ऍड्रेसवर 200 जोडा.

(ड) बेरीजचे स्थान सुरू करण्यासाठी 200 बाइट मेमोरी नियुक्त करते.

प्रश्न 4.4 असेम्बली लॉगवेज ही _____ प्रत्येक इंस्ट्रक्शन सेट आर्किटेक्चर (ISA) आणि असेम्बलरसाठी विशिष्ट प्रोग्रामिंग लॉगवेज.

(अ) उच्च-स्तरीय (ब) मध्यम-स्तरीय (क) निम्न-स्तरीय (ड) वस्तुनिष्ठ

प्रश्न 4.5 आपण एनएएसएम असेम्बली भाषेत टिप्पण्या कशा वापरता?

अ) अर्धविराम (;) (ब) टक्के चिन्ह (%) (क) फॉरवर्ड स्लॅश (/) (ड) अल्पविराम (,)

प्रश्न 4.6 असेम्बलर स्वयंचलितपणे असेम्बली लॉगवेज प्रोग्रामचे

(अ) बाइट कोड (ब) बायनरी नंबर (क) विशेष वर्ण (ड) मशीन इंस्ट्रक्शनच्या क्रमाने अनुवादित करते.

प्रश्न 4.7 खालीलपैकी कोणत्या असेम्बलरचे नाव आहे?

(अ) MASM (ii) TASM (iii) GAS (iv) NASM (a) (i) (iii) आणि (iv)

(ब) (i) (ii) आणि (iii)

(क) फक्त (i) आणि (ii)

(ड) सर्व

प्रश्न 4.8 जेव्हा डारेक्टिवचा विशिष्ट क्रम प्रोग्राममधील वेगवेगळ्या बिंदूंमध्ये पुनरावृत्ती केला जातो.

प्रोग्राममधील इंस्ट्रक्शनचे हे अनुक्रम "सबप्रोग्राम" म्हणून लिहिले जाऊ शकतात ज्याला a म्हणतात. _____

(अ) डाइरेक्टिव्ह (ब) प्रोग्रॅम (क) प्रोसिजर (ड) माक्रो

प्रश्न 4.9 एसेम्बलर ऑब्जेक्ट प्रोग्राम सेटयित करते _____

अ) कॅशे मेमोरी (ब) प्राथमिक मेमोरी (क) दुय्यम मेमोरी (ड) रॉम मेमोरी

प्रश्न 4.10 प्रोसीजरमध्ये, इंस्ट्रक्शन इंस्ट्रक्शनचा क्रम कार्यान्वित करते आणि इंस्ट्रक्शन प्रोग्राममधील पुढील इंस्ट्रक्शनवर अंमलबजावणी परत करते.

(अ) कॉल, रेट (ब) स्टार्ट, एंड (क) स्टार्ट, रिटर्न (ड) डू, वाइल

प्रश्न 4.11 NASM प्रोग्राममध्ये सिस्टम एक्झिटसाठी कोणता सिस्टम कॉल वापरला जातो?

(अ) int 0x80 (ब) mov eax, 4 (क) mov eax, 1 (ड) mov ebx, 2

प्रश्न 4.12 एक्झिक्युशनसाठी ऑब्जेक्ट कोड मेमोरीमध्ये आणते.

(अ) लिंकर (ब) एक्सेट्रॅक्टर (क) इटर (ड) लोडर

प्रश्न 4.13 NASM असेंबलरमध्ये कोणता विभाग वापरत नाही?

(अ) डेटा (ब) टेक्स्ट (क) बीएसएस (ड) कोड

बहुपर्यायी प्रश्नांचे उत्तरे

4.1	(क)	4.2	(ब)	4.3	(ब)	4.4	(क)	4.5	(अ)	4.6	(ड)
4.7	(ड)	4.8	(क)	4.9	(क)	4.10	(अ)	4.11	(क)	4.12	(ड)
4.13	(ड)										

लघु आणि दीर्घ उत्तर प्रकार

प्रश्न श्रेणी-I

प्रश्न 4.1 असेम्बलर (assembler) म्हणजे काय?

प्रश्न 4.2 चार लोकप्रिय असेंबलर्सची यादी करा.

प्रश्न 4.3 असेम्बली लॉगवेज प्रोग्रामिंग म्हणजे काय?

प्रश्न 4.4 मशीन इंस्ट्रक्शन काय आहेत?

प्रश्न 4.5 असेंबलर, लोडर आणि लिंकरची व्याख्या करा.

प्रश्न 4.6 NASM मध्ये वापरल्या जाणार्या लॉजिकल ऑपरेशन्सची संख्या सूचीबद्ध करा.

प्रश्न 4.7 NASM मध्ये अरिथमेटिक ऑपरेशन्सची यादी करा.

प्रश्न 4.8 सलग मेमोरी ऍड्रेसमध्ये सेटयित केलेल्या अरेच्या घटकांमध्ये प्रवेश करण्यासाठी कोणते अँड्रेसिंग मोड श्रेयस्कर आहे?

प्रश्न 4.9: कंडिशनल आणि अन कंडिशनल ब्रांच मध्ये काय फरक आहे?

प्रश्न 4.10 दोन स्ट्रिंग मानिपुलेशन ऑपरेशन्सचे उदाहरण द्या.

प्रश्न 4.11 Linux 64-bit संगणकात assembling language प्रोग्राम चालवण्यासाठी तुम्ही कोणती कमांड वापरता?

प्रश्न 4.12 उच्च आणि निम्न स्तरावरील लॉगवेज मध्ये काय फरक आहे?

प्रश्न 4.13 सिस्टम कॉल 1 आणि 4 चा अर्थ काय आहे?

प्रश्न 4.14 NASM मध्ये तुम्ही लेबल आणि टिप्पण्या कशा वापरता?

प्रश्न 4.15 NASM मध्ये Define byte (DB) ची उपयुक्तता स्पष्ट करा.

प्रश्न श्रेणी-II

प्रश्न 4.16 NASM च्या स्थापनेच्या पायऱ्या समजावून सांगा.

प्रश्न 4.17 NASM असेम्बली लॉगवेज कार्यक्रमांमध्ये कार्यपद्धती आणि माक्रो कशा परिभाषित केल्या जातात? माक्रो आणि कार्यपद्धतींमध्ये काय फरक आहे?

प्रश्न 4.18 नेस्टेड प्रक्रिया म्हणजे काय? प्रक्रिया आणि माक्रोचे फायदे आणि तोटे सूचीबद्ध करा.

प्रश्न 4.19 असेम्बलर डायरेक्टिव्ह (assembler directive) म्हणजे काय? उदाहरणांसह समजावून सांगा.

प्रश्न 4.20 NASM डिरेक्टरीमध्ये उपलब्ध असलेल्या सिस्टम कॉलची यादी करा.

प्रश्न 4.21 CALL आणि RET च्या प्रक्रिया स्पष्ट करा.

संख्यात्मक प्रॉब्लेम्स

प्रश्न 4.22 असेम्बली लॉगवेज प्रोग्राम वापरून टर्मिनलवर आपले नाव आणि पालकांचे नाव प्रदर्शित करा (ALP).

प्रश्न 4.23: 40 मधील 12 वजा करण्यासाठी ALP लिहा. त्यानंतर, निकालावर, NOT ऑपरेशन कार्यान्वित करा आणि निकाल स्क्रीनवर दाखवा.

प्रश्न 4.24 माक्रो वापरून पूर्णांक 3 आणि 2 वर AND आणि OR लॉजिकल ऑपरेशन्ससाठी ALP लिहा.

प्रश्न 4.25: पूर्णांक 5 आणि 2 मधील ALP आणि ऑपरेशन लिहा. नंतर, अंतिम परिणाम मिळविण्यासाठी व्हॅल्यूला 4 ने गुणाकार करा. अंतिम परिणाम म्हणून टर्मिनल काय दर्शविते?

प्रश्न 4.26 1 6 ला 2 ने भाग देण्यासाठी ALP लिहा. टर्मिनलवर निकाल दाखवा.

प्रश्न 4.27 ब्रांच इंस्ट्रक्शन वापरून ए. एल. पी. लिहा. जर संख्या 5 पेक्षा मोठी असेल तर 1 आणि 2 चे लॉजिकल OR करा; अन्यथा, त्याच संख्यांवर लॉजिकल AND करा. टर्मिनलवर निकाल दाखवा.

प्रश्न 4.28 एक ALP लिहा जे इनपुट घटक अरेमध्ये सेटयित करेल आणि टर्मिनलवर प्रदर्शित करेल. प्रोग्रॅममध्ये त्यांचा वारंवार वापर सुलभ करण्यासाठी प्रणाली कॉल प्रक्रिया म्हणून लिहिले गेले पाहिजेत.

प्रात्यक्षिक:

उद्देश: बेरीज, वजाबाकी, गुणाकार आणि भागाकार कार्य करू शकणारे एरिथमेटिक गणकयंत्राची रचना करण्यासाठी असेम्बली लॉगवेज प्रोग्राम लिहा.

NASM असेंबलर वापरून $(a + b) * 3 - c/d$ या समीकरणांची गणना करा. जेथे $a = 8$, $b = 2$, $c = 4$, आणि $d = 1$

साधने: NASM असेंबलर [1]

सिद्धांत: NASM असेंबलरचे तपशील या अध्यायात आधीच चर्चा केलेले आहेत.

प्रोग्रामिंग शिकण्यासाठी

प्रक्रिया: संगणकावर लिनक्स 64-बिट स्थापित करण्यासाठी तपशीलवार इंस्ट्रक्शन या अध्यायात आधीच समाविष्ट केल्या आहेत.

अधिक जाणून घ्या:**भारतीयांनी केलेले नवकल्पना**

थिंक टँक टीमचे नेते आणि संगणक शास्त्रज्ञ आणि शोधक म्हणून प्रणव मिस्त्री हे सिक्स्थसेन्समधील त्यांच्या योगदानासाठी प्रसिद्ध आहेत.

सिक्स्थसेन्स हे एक नाविन्यपूर्ण ऑगमेंटेड रिअॅलिटी (AR) तंत्रज्ञान आहे जे वापरकर्त्यांना त्यांची बोटे हलवून आणि हवेत त्यांच्या बोटांच्या टोकांनी त्यावर कन्ट्रोल ठेवून भिंतीवर पडदा प्रक्षेपित करण्यास सक्षम करते.

डारेक्टिव सेट वास्तुकलेचा इतिहास

ARM आणि x86 ही लोकप्रिय डारेक्टिव सेट रचना आहेत. ARM हे 'प्रगत RISC मशीन्स' चे संक्षिप्त रूप आहे. हा युनायटेड किंगडममधील केंब्रिज येथे स्थित एक सुप्रसिद्ध व्यवसाय आहे. 2012 मध्ये, सुमारे 90% मोबाइल डिव्हाइसेस एआरएम-आधारित प्रोसेसरद्वारे समर्थित होते. इंटेल आणि ए. एम. डी. x86 प्रोसेसर वापरतात, जे डेस्कटॉप आणि लॅपटॉप बाजारातील नव्वद टक्क्यांहून अधिक वर्चस्व गाजवतात. ARM चा डारेक्टिव सेट RISC आहे, तर x86 चा डारेक्टिव सेट CISC आहे.



स्कॅन करा

ARM आणि x86 बदल
अधिक जाणून घेण्यासाठी

भारतीय योग आणि प्राणायाम

प्राणायाम म्हणजे श्वासोच्छ्वासावर कन्ट्रोल ठेवण्याचा सराव. 'प्राण' म्हणजे एखाद्या व्यक्तीचा श्वास किंवा प्राणशक्ती. "अयमा" तंत्र अभ्यासकाला प्राण किंवा जीवन देणारी प्राण ऊर्जा व्यवस्थापित करण्यास अनुमती देते [2].

प्राणायाम म्हणजे शरीर आणि मन यांना जोडण्यासाठी श्वासोच्छ्वासावर कन्ट्रोल ठेवण्याचा सराव. प्राणायामाचे वर्णन पतंजली चेतनेच्या उच्च अवस्था प्राप्त करण्याची एक पद्धत म्हणून करते. हा योगाचा एक आवश्यक घटक

आहे, जो शारीरिक आणि मानसिक आरोग्याला चालना देणारा सराव आहे. "प्राण" म्हणजे संस्कृतमध्ये जीवन ऊर्जा आणि "यम" म्हणजे कन्ट्रोल [3].

प्राणायामामध्ये श्वासोच्छ्वासाचे विविध व्यायाम आणि नमुने समाविष्ट असतात. तुम्ही जाणूनबुजून श्वास आत घेता, सोडता आणि तुमचा श्वास एका विशिष्ट क्रमाने राखून ठेवता. प्राणायाम हा शारीरिक मुद्रा (आसन) आणि ध्यान यासारख्या इतर योग पद्धतींच्या संयोगाने वापरला जातो. (dhyana). एकत्रितपणे, या पद्धती योगाच्या अनेक फायद्यांसाठी जबाबदार आहेत. प्राणायाम ही प्रत्येक श्वासाची आणि धरणाची वेळ, कालावधी आणि वारंवारता नियंत्रित करण्याची प्राचीन प्रथा आहे. हे विषारी पदार्थ काढून टाकताना शरीराला ऑक्सिजन पुरवते. हे असंख्य प्राणांच्या कृतींमध्ये समतोल साधते, परिणामी शरीर आणि मन निरोगी होते.

समकालीन जीवनशैलीच्या संदर्भात प्राणायामच्या फायद्यांचा वैज्ञानिकदृष्ट्या अभ्यास केला गेला आहे. हजारो वर्षांपूर्वी भारत हे योगाचे जन्मस्थान होते. योगाच्या वैज्ञानिक अभ्यासाच्या विस्ताराबरोबरच त्याच्या उपचारात्मक पैलूंचाही शोध घेतला जात आहे. तणाव कमी करणे, झोपेची गुणवत्ता वाढवणे, सजगता वाढवणे, उच्च रक्तदाब कमी करणे, फुफ्फुस आणि मेंदूचे कार्य वाढवणे, पाचक प्रणाली, रोगप्रतिकारक शक्ती वाढवणे आणि श्वसन प्रणाली बळकट करणे यासारख्या विविध मार्गांनी योग आणि प्राणायाम आरोग्यासाठी फायदेशीर ठरू शकतात.

संदर्भ आणि सुचविलेले वाचन

[1] मुहम्मद यझार वाय, एनएएसएमचा परिचय.

<https://usermanual.wiki/Document/NASM20Manual.116>

4426225/view (last accessed: May 15, 2023)

[2] प्राणायाम म्हणजे काय आणि त्याचे प्रकार आणि तंत्रे?

<https://www.artofliving.org/in-en/yoga/what-ispranayama-and-istypes-techniques> (last accessed May 14, 2023)

[3] योग आणि प्राणायाम यांचे आरोग्यावर होणारे परिणाम: एक अत्याधुनिक पुनरावलोकन.

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3415184/> / (अखेरचा प्रवेश: 14 मे 2023)

[4] एनपीटीईएल अभ्यासक्रम प्रा. जानकीरमन विराराघवन, सी प्रोग्रामिंग आणि असेम्बली लॉगवेज, आयआयटी मद्रास, 2019. <https://nptel.ac.in/courses/108105102> (last accessed: May, 2023)



स्कॅन करा

असेम्बली प्रोग्राम्ससह C
लायब्ररीचा वापर
शिकण्यासाठी

5. मेमोरी आणि डिजिटल इंटरफेसिंग

युनिट ची वैशिष्ट्ये

या युनिटमध्ये खालील पैलूंवर चर्चा केली आहे:

- मेमोरी अँड्रेसिंग आणि अँड्रेस डीकोडिंग;
- इंटरफेसिंग रॅम, रॉम, ईपीरॉम;
- प्रोग्रामेबल पेरिफेरल इंटरफेस आणि ऑपरेशन मोड;
- प्रोसेसरशी इंटरफेस करण्याची विविध तंत्रे;
- कीबोर्ड आणि डिस्प्ले इंटरफेसिंग;

अधिक जिज्ञासा आणि सर्जनशीलता वाढवण्यासाठी आणि समस्या सोडवण्याची कौशल्ये वाढवण्याच्या उद्देशाने विषयांचे व्यावहारिक अनुप्रयोग सादर केले जातात. ब्लूमच्या वर्गीकरणशास्त्राच्या खालच्या आणि वरच्या स्तरांनुसार दोन श्रेणींमध्ये चिन्हांकित केलेल्या मोठ्या संख्येने बहुपर्यायी प्रश्न आणि लहान आणि दीर्घ उत्तराच्या प्रश्नाव्यतिरिक्त, युनिट संख्यात्मक समस्यांच्या स्वरूपात सराव असाइनमेंट, संदर्भाची यादी आणि सुचविलेले वाचन प्रदान करते.

हे लक्षात घेणे महत्वाचे आहे की स्वारस्य असलेल्या विविध विषयांवरील अधिक माहितीसाठी स्कॅन केले जाऊ शकणारे अनेक क्यूआर कोड वेगवेगळ्या भागांमध्ये समाविष्ट केले गेले आहेत आणि आवश्यक आधार डेटा प्राप्त करण्यासाठी वापरले जाऊ शकतात.

सामग्रीवर आधारित संबंधित व्यावहारिक नंतर विषयावरील "अधिक जाणून घ्या" विभाग आहे. हा विभाग काळजीपूर्वक अशा प्रकारे तयार करण्यात आला आहे की त्यात असलेली पूरक माहिती पुस्तकाच्या वाचकांसाठी मौल्यवान असेल. हा विभाग प्रामुख्याने दैनंदिन जीवनात ऊर्जावान, केंद्रित आणि तणावमुक्त राहण्यासाठी संगणक प्रणाली संघटना आणि भारतीय ध्यान पद्धतींच्या विकासासाठी भारतीय नवसंशोधकांच्या योगदानावर लक्ष केंद्रित करतो.

तर्क

या अध्यायात अनेक मेमोरी तंत्रज्ञान आणि साठवण क्षमता आणि डेटा ट्रान्सफरच्या गतीच्या मागण्या पूर्ण करण्यासाठी ते कसे विकसित झाले आहेत यावर चर्चा केली आहे. सेमीकंडक्टर तंत्रज्ञानाच्या प्रगतीमुळे मेमोरीचा वेग आणि क्षमतेत लक्षणीय सुधारणा झाल्या आहेत, तसेच प्रति बिट किंमतीत लक्षणीय घट झाली आहे. विविध मेमोरी तंत्रज्ञानाचे वर्गीकरण विविध अनुप्रयोगांसाठी भिन्न तंत्रज्ञानाच्या आवश्यकता समजून घेण्यास मदत करते. मेमोरी-टोकेश डेटा मापिंग तंत्रे देखील समाविष्ट आहेत. शिवाय, दुय्यम मेमोरीबद्दल चर्चा केल्याने तुम्हाला वेग, क्षमता आणि तंत्रज्ञानाची प्राथमिक मेमोरीशी तुलना करता येते. या अध्यायात मेमोरी आणि आय/ओ इंटरफेस

देखील समाविष्ट आहेत. एक्सटर्नल उपकरणे, प्रोसेसर, कीबोर्ड, डिस्प्ले इत्यादींसाठी असंख्य इंटरफेस तंत्रांचे सखोल वर्णन केले आहे.

पूर्व-आवश्यकता

संगणक प्रणाली संघटना (Unit-I)

डिजिटल इलेक्ट्रॉनिक्स: संख्या प्रणाली-बायनरी, ऑक्टल आणि हेक्साडेसिमल (Polytechnic Engineering)

युनिट निष्पत्ती

या युनिटच्या परिणामांची यादी खालीलप्रमाणे आहे:

U5-O1: प्रोसेसरसाठी विविध मेमोरी इंटरफेसच्या भूमिकेचे वर्णन करा.

U5-O2: मेमोरी तंत्रज्ञानाच्या भूमिकेचे वर्णन करा. मेमोरी रचनेत RAM, ROM, EPROM.

U5-O3: प्रोग्रामेबल एक्सटर्नल इंटरफेस आणि ऑपरेशनच्या विविध पद्धती स्पष्ट करा.

U5-O4: प्रोसेसरला इंटरफेस करण्याची विविध तंत्रे स्पष्ट करा.

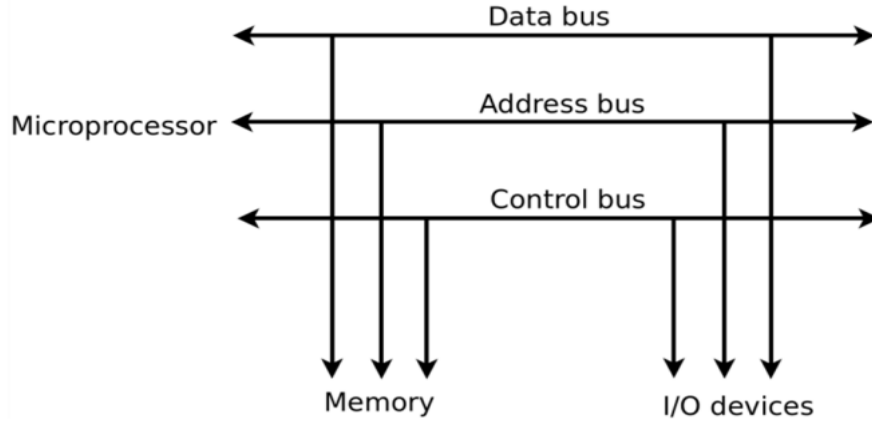
U5-O5: कळफलक आणि प्रदर्शनासह परस्परसंबंध स्पष्ट करा.

युनिट -5 निष्पत्ती	निष्पत्ती सह अपेक्षित मार्पिंग				
	4- कमकुवत मार्पिंग, 2- मध्यम मार्पिंग, 3- मजबूत मार्पिंग				
	CO-1	CO-2	CO-3	CO-4	CO-5
U5-01	3	3	3	2	2
U5-02	3	3	2	3	2
U5-03	2	1	3	1	2
U5-04	3	3	2	1	1
U5-05	3	3	3	1	3

5.1 परिचय

इंटरफेस हा दोन घटकांमधील संवादाचा मार्ग आहे. आकृती 5.1 मध्ये दर्शविल्याप्रमाणे इंटरफेसिंग दोन प्रकारचे असतात, मेमोरी इंटरफेसिंग आणि आय/ओ (इनपुट/आउटपुट) इंटरफेसिंग. मेमोरी इंटरफेसिंगमध्ये, इंस्ट्रक्शन अंमलबजावणीदरम्यान, प्रोसेसर वाचन आणि लेखन कार्यासाठी मेमोरीशी संवाद साधतात. प्रोसेसर मेमोरीतून डेटा वाचतो किंवा मेमोरीमध्ये डेटा लिहितो.

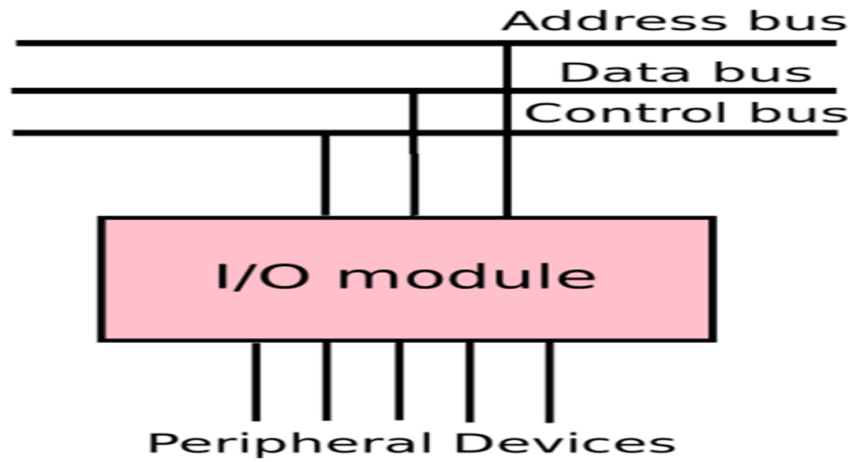
रीड/राइट कार्य करण्यापूर्वी, मायक्रोप्रोसेसर मेमोरीवर रीड/राइट कंट्रोल संकेत पाठवतो. आय/ओ इंटरफेसिंगमध्ये, प्रोसेसर आय/ओ उपकरणांशी आय/ओ मॉड्यूलसद्वारे संवाद साधतात ज्यामध्ये एक्सटर्नल उपकरणे आणि सिस्टम बस [1] दरम्यान संप्रेषण करण्यासाठी तर्क असतो.



आकृती. 5.1: मेमोरी आणि आय/ओ इंटरफेस

आकृती 5.2 दर्शविते की एक्सटर्नल आणि प्रणाली बस दरम्यान एक महत्वाचा इंटरफेस म्हणून I/O मॉड्यूल आवश्यक आहे कारण

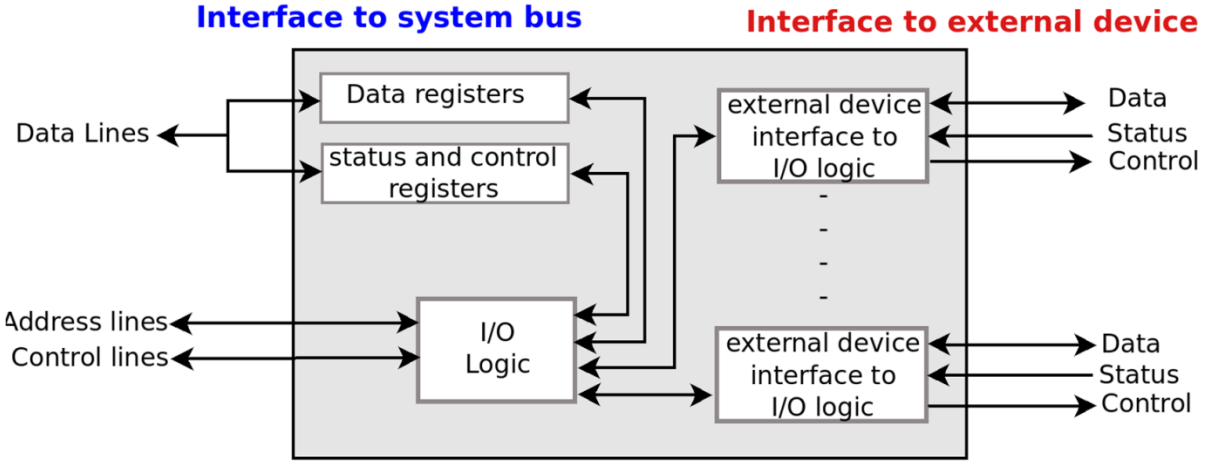
- एक्सटर्नलचे अनेक वेगवेगळे प्रकार अस्तित्वात आहेत, प्रत्येकाची स्वतःची कार्य करण्याची अनोखी पद्धत आहे. सर्व उपकरणे एकाच केंद्रीय प्रक्रिया युनिटद्वारे (सीपीयू) नियंत्रित केली जाऊ शकत नाहीत कारण प्रोसेसरमध्ये आवश्यक कार्यक्षमता समाविष्ट करणे अव्यवहार्य असेल.
- एक्सटर्नल डेटा ट्रान्स्फर दर मुख्य मेमोरी/सीपीयू पेक्षा कमी आहे. या कारणास्तव, हाय-स्पीड सिस्टम बसद्वारे एक्सटर्नल उपकरणासह संवाद साधणे शक्य नाही.



आकृती. 5.2 एक्सटर्नल उपकरणे

- अनेक एक्सटर्नल सेट विविध स्वरूपांमध्ये आणि यजमान संगणकाशी विसंगत असलेल्या भिन्न वर्ड लांबीसह माहिती सेटयित आणि ट्रान्स्फर करतात.

I/O मॉड्यूलची ब्लॉक आकृती आकृती 5.3 मध्ये दर्शविली आहे. डाव्या बाजूचा इंटरफेस प्रणाली बसशी जोडलेला आहे आणि उजव्या बाजूचा इंटरफेस कीबोर्ड, माऊस, बाह्य मेमोरी इत्यादी बाह्य उपकरणांशी जोडलेला आहे.



आकृती. 5.3: I/O मॉड्यूलची ब्लॉक आकृती

अंतर्गत संसाधने आणि बाह्य उपकरणांमधील माहितीच्या प्रवाहाच्या वेळेचे समन्वय साधण्यासाठी कंट्रोल संकेतांची आवश्यकता असते. यंत्राची स्थिती जाणून घेण्यासाठी प्रोसेसर आय/ओ मॉड्यूलची चौकशी करतो. डिव्हाइस तयार असल्यास आय/ओ मॉड्यूलद्वारे डेटा प्रोसेसरकडे ट्रान्सफर केला जातो. आय/ओ मॉड्यूल बाह्य उपकरणातून डेटा प्राप्त करते.

5.2 मेमोरी प्रकार आणि वैशिष्ट्ये

या विभागात विविध प्रकारच्या मेमोरी समाविष्ट आहेत ज्या संगणक प्रणाली वापरू शकते. मेमोरीचे प्रत्येक वैशिष्ट्य आणि कार्य तत्त्व यावर सखोल चर्चा केली आहे.

5.2.1 मेमोरीचे प्रकार

वापरलेले परिपथ रचना तंत्र, तात्पुरत्या किंवा कायमस्वरूपी साठवण क्षमतेचे प्रमाण, प्रणालीतील साईझ, कॉस्ट आणि लोकेशन यावर आधारित मेमोरीचे अनेक प्रकारांमध्ये वर्गीकरण केले जाते.

1. रजिस्टर

सामान्य उद्देश रजिस्टर, विभाग रजिस्टर, डारेक्टिवक रजिस्टर आणि डारेक्टिवक रजिस्टर यासारख्या रजिस्टर सीपीयू चा भाग आहेत. रजिस्टरची साठवण क्षमता बाइटमध्ये परिभाषित केली जाते. रजिस्टर हे सर्वात जलद प्रवेश मेमोरी घटक आहेत.

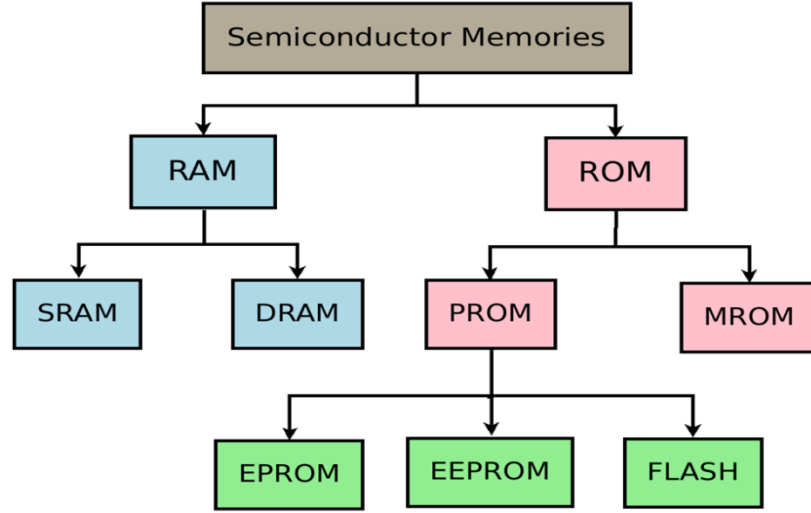
2. सेमीकंडक्टर मेमोरी

सेमीकंडक्टर मेमोरी यादृच्छिक प्रवेश मेमोरी आणि केवळ वाचन मेमोरीमध्ये वर्गीकृत केल्या जातात.



स्कॅन करा

SRAM आणि DRAM सर्किट डिझाइनसाठी



आकृती. 5.4 सेमीकंडक्टर मेमोरी वर्गीकरण

5.2.2 रँडम ऍक्सेस मेमोरी (RAM)

रँडम ऍक्सेस मेमोरी (रँम) चा वापर डेटा आणि प्रोग्राम वापरण्यापूर्वी तात्पुरते संचयित करण्यासाठी केला जातो. भौतिक मेमोरीतील विशिष्ट ऍड्रेसवर डेटा रीड/राइट जाऊ शकतो. रँममध्ये मल्टिप्लेक्सिंग आणि डिमल्टिप्लेक्सिंग सर्किट्स आहेत जे डेटा लाइनला ऍड्रेस लाइन शी जोडतात, ज्यामुळे वैयक्तिक डेटा घटकांचे वाचन आणि लेखन शक्य होते [2]. रँमच्या मेमोरी सेल मूलतः इलेक्ट्रॉनिक सर्किट्स आहेत ज्या संगणकासाठी डेटा संचयित करू शकतात. सेट आणि रीसेट लॉजिक मेमोरी सेलमध्ये डेटा सेटयित करण्यासाठी वापरले जाते, "सेट" म्हणजे "1" (0.5 व्होल्टपेक्षा जास्त) आणि "रीसेट" म्हणजे "लॉजिक 0" (below 0.5 volts). ती एक अस्थिर मेमोरी आहे. रँम मेमोरीचे डायनॅमिक रँम (DRAM) आणि स्टॅटिक रँम (SRAM) मध्ये वर्गीकरण केले जाते. दोन्ही प्रकारच्या मेमोरी अस्थिर मेमोरी असतात. जर पॉवर ऑफ झाली तर मेमोरीमध्ये डेटा नष्ट होतो.

DRAM ला प्राथमिक किंवा मुख्य मेमोरी असेही म्हणतात आणि त्याची साठवण क्षमता MB ते GB असते. प्रत्येक DRAM मेमोरी सेल केवळ एक ट्रान्झिस्टर आणि एक कॅपेसिटरने बनलेला असतो. डेटा कॅपेसिटरमध्ये संग्रहित केला जातो आणि प्रत्येक कॅपेसिटरची चार्ज पातळी बिट लॉजिकल 1 किंवा 0 आहे की नाही हे ठरवते. कारण कॅपेसिटर डिस्चार्ज झाल्यास माहिती नष्ट होऊ शकते, कॅपेसिटर चार्ज असताना मेमोरी सेल एकतर 0 किंवा 1 संचयित करते. या मेमोरी सेल नियमित अंतराने आपोआप चार्ज होतात. कारण MOS कॅपेसिटर वापरून DRAM कार्यान्वित केले जाते. त्यामुळे डेटा संचयित करण्यासाठी भरपूर शक्तीची आवश्यकता असते.

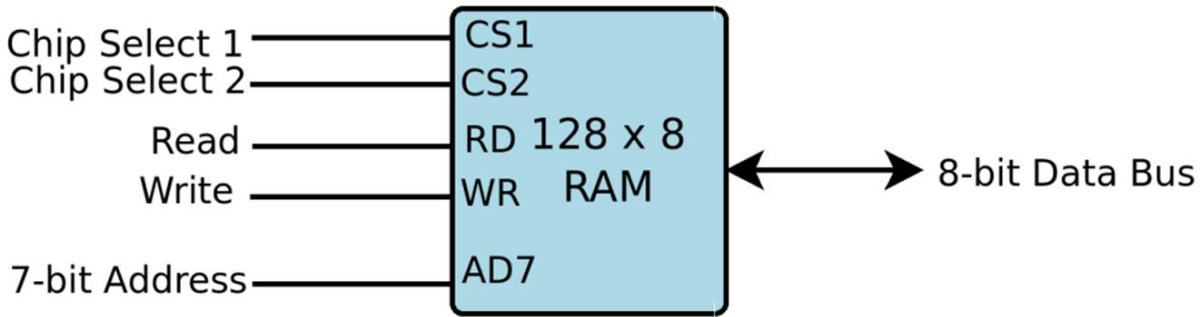
तर, SRAM चा वापर कॅशे मेमोरी डिझाइन करण्यासाठी केला जातो. प्रत्येक SRAM सेल सर्किटला सहा ट्रान्झिस्टरची आवश्यकता असते, चार बिट संचयित करण्यासाठी आणि दोन सेलमध्ये प्रवेश नियंत्रित करण्यासाठी. SRAM मध्ये, डेटा नियमितपणे रीफ्रेश करण्याची आवश्यकता नाही [3]. SRAM मेमोरी सेल सामान्यतः MOSFET- आधारित फ्लिप-फ्लॉप सर्किट्स असतात. कॅशे मेमोरीमध्ये KB ते MB पर्यंत स्टोरेज स्पेस असते, जी

रजिस्टर स्टोरेज स्पेसपेक्षा जास्त असते. अलीकडे वापरलेला डेटा कॅशे मेमोरीमध्ये संग्रहित केला जातो. रजिस्टर प्रवेश वेळेच्या तुलनेत, कॅशे मेमोरी प्रवेश वेळ जास्त आहे.

तक्ता 5.1: SRAM आणि DRAM मधील फरक

SRAM	DRAM
यांचा प्रवेश वेळ कमी आहे, त्यामुळे SRAM हा DRAM पेक्षा वेगवान आहे.	यांचा प्रवेश वेळ जास्त आहे, त्यामुळे DRAM हा SRAM पेक्षा संथ आहे.
हे DRAM पेक्षा महाग आहे.	हे SRAM पेक्षा कमी खर्चिक आहे.
हे सर्व वेळ चालू ठेवणे आवश्यक आहे, त्यामुळे SRAM अधिक शक्ती वापरते.	DRAM कमी शक्ती वापरते कारण ते कॅपेसिटरमध्ये माहिती साठवते.

SRAM आणि DRAM मेमोरीची वैशिष्ट्ये टेबल 5.1 मध्ये तुलना केली आहेत. वीज बंद असल्यास स्टॅटिक रॅममध्ये साठवलेला डेटा नष्ट होतो. SRAM हे DRAM पेक्षा लक्षणीयरीत्या वेगवान आहे. हे DRAM पेक्षा अधिक महाग आहे कारण प्रत्येक सेलला सहा ट्रान्झिस्टरची आवश्यकता असते.



आकृती. 5.5: 7-अॅड्रेस लाईन्स आणि 8-द्विदिशात्मक डेटा लाईन्स असलेली एक विशिष्ट रॅम चिप

$2^n \times d$ RAM चिपमध्ये n अॅड्रेस लाईन्स आणि d बायडायरेक्शनल डेटा लाईन्स असतात. आकृती 5.5 रॅम चिपची रचना दर्शविते, ज्यामध्ये दोन निवड लाइन (CS1 आणि CS2) सात अॅड्रेस लाइन (AD7) आणि 8-बिट बायडायरेक्शनल डेटा लाइन समाविष्ट आहेत. निवडलेल्या बिट व्हॅल्यूच्या आधारे रीड/राइट कार्य सक्रिय केले जाते. रॅम चिपमध्ये 128 (2^7) मेमोरी स्थाने आहेत जी 7-बिट अॅड्रेससह दर्शविली जाऊ शकतात आणि प्रत्येक मेमोरी स्थानाची क्षमता 8 बिट आहे. प्रोसेसर रॅम चिपमधून 8 बिट डेटा मिळवू शकतो किंवा एका वेळी रॅम चिपमध्ये 8 बिट डेटा लिहू शकतो. रॅम मेमोरी हा तात्पुरत्या मेमोरीचा एक प्रकार आहे.

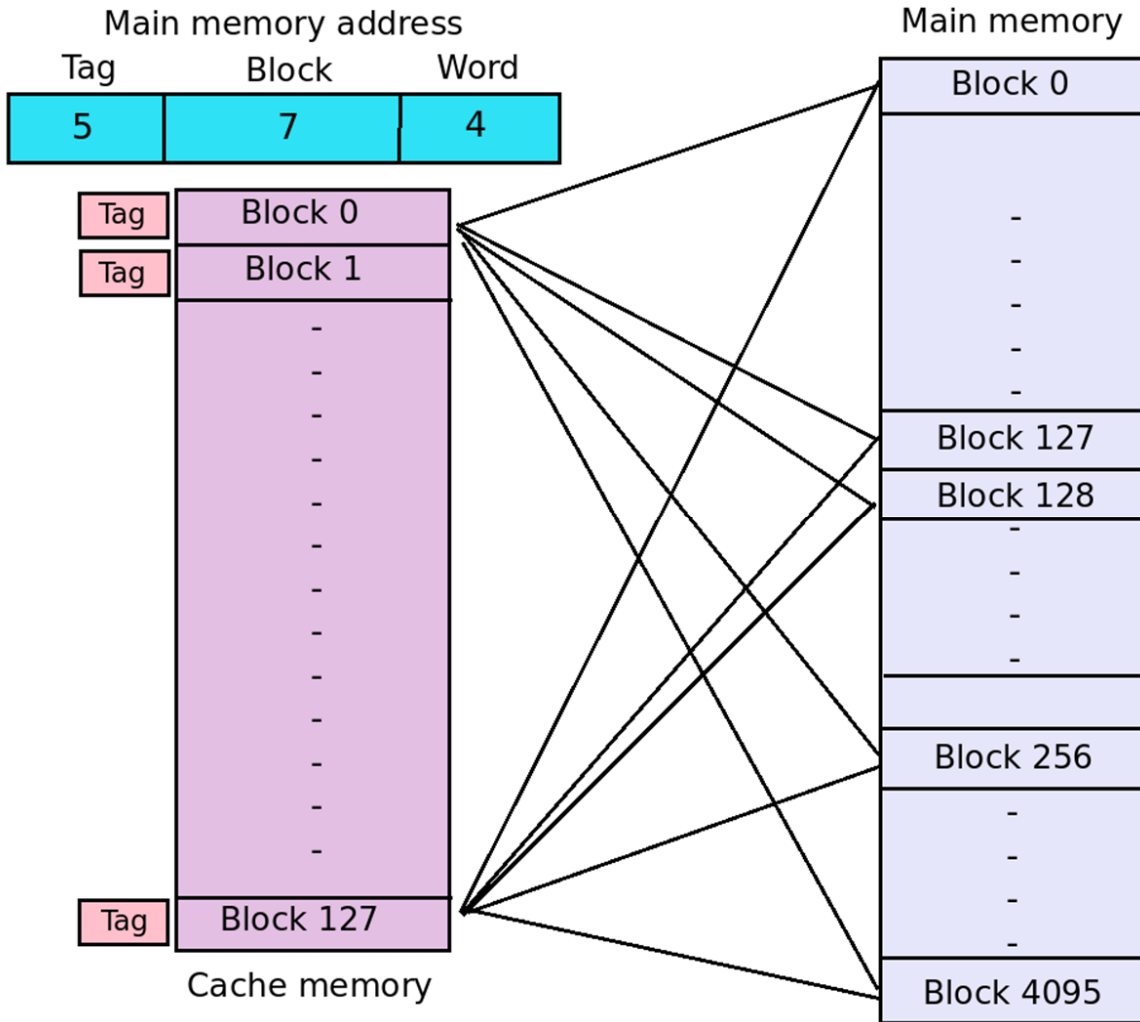
5.2.3 कॅशे मेमोरी मारपिंग तंत्र

कॅशे मेमोरी मुळे मुख्य मेमोरी जलद गतीने कार्य करते हे संगणकाला समजते. लूप्स, नेस्टेड लूप्स आणि फंक्शन्स ही इंस्ट्रक्शनची उदाहरणे आहेत जी प्रोग्राममध्ये एकमेकांना वारंवार कॉल करतात. प्रोग्रॅमच्या विशिष्ट फील्डतील अनेक इंस्ट्रक्शन कालांतराने वारंवार अंमलात आणल्या जातात. याला 'लोकॅलिटी ऑफ रेफरेंस' असे म्हणतात. याचे दोन प्रकारात वर्गीकरण केले जाते: स्पेटिअल लोकॅलिटी आणि टेम्पोरल लोकॅलिटी. अरेसारख्या

तुलनेने जवळच्या साठवण ऍड्रेसमधील डेटा घटकांचा वापर हा स्थानिक स्थान म्हणून ओळखला जातो. तात्पुरत्या परिसरातील लोकांचा असा विश्वास आहे की नुकत्याच अंमलात आणलेल्या इंस्ट्रक्शन पुन्हा अंमलात येण्याची शक्यता जास्त असते.

कॅशेचे दोन प्रकारात वर्गीकरण केले जाते: 'राइट-थ्रू' कॅशे आणि 'राइटबॅक' कॅशे. "राइट-थ्रू" कॅशे एकाच वेळी कॅशे आणि मुख्य मेमोरी ऍड्रेसमध्ये बदल करतात. राइटबॅक कॅशेमध्ये, फक्त कॅशेचे स्थान अद्ययावत केले जाते आणि संबंधित फ्लॅग बिट अद्ययावत म्हणून चिन्हांकित केले जातात; हा बिट सामान्यतः डर्टी किंवा मॉडिफाइड बिट म्हणून ओळखला जातो. जेव्हा हा ब्लॉक काढून टाकला जातो, तेव्हा तो मेमोरी मध्ये साठवला जातो. ही पद्धत वापरणाऱ्या कॅशेना 'राइटबॅक' किंवा 'कॉपी बॅक' कॅशे म्हणतात. जेव्हा नवीन कॅशे ब्लॉक साठी जागा तयार करण्यासाठी कॅशेमधून कॅशे ब्लॉक काढला जातो तेव्हा ही प्रक्रिया घडते.

मुख्य मेमोरीचे ब्लॉक कॅशे मेमोरीमध्ये तीन वेगवेगळ्या प्रकारे माप केले जाऊ शकतात: डीरेक्ट मॅपिंग, फुल्ली-असोसिएटिव्ह मॅपिंग आणि सेट-असोसिएटिव्ह मॅपिंग. या मॅपिंग योजनांचा कॅशे हिट रेशोवर आणि प्रणालीच्या कामगिरीवर लक्षणीय परिणाम होतो. प्रत्येक मॅपिंग चे फायदे आणि तोटे आहेत. या मॅपिंग पद्धतींवर खाली अधिक तपशीलवार चर्चा केली आहे.



आकृती 5.6: सामान्यतः 4096 मुख्य मेमोरी ब्लॉक्सचे 128 कॅशे ब्लॉक्सचे थेट कॅशे ब्लॉक मॅपिंग

1. डरेक्ट मॅपिंग

कॅशे मेमोरीची क्षमता मुख्य मेमोरीपेक्षा कमी असते. कॅशे ऍड्रेस C बिट्स वापरतो, जो मुख्य मेमोरीतील N बिट्सपेक्षा कमी असतो. समजा मुख्य मेमोरीची क्षमता $2N$ बाइटची आहे, जी B ब्लॉकमध्ये विभागली गेली आहे. (0 to B-1 integer value). प्रत्येक ब्लॉक डेटाचे D बाइट संचयित करतो. कॅशे मिस झाल्यास, आवश्यक ब्लॉक 2^C कॅशे मेमोरीमध्ये मॅपिंग केला जातो. मुख्य मेमोरीचा ब्लॉक j (जेथे $0 \leq j < B$) म्हणून कॅशे ब्लॉक नंबरवर मॅपिंग केला जातो ज्याची गणना म्हणून केली जाते.

कॅशे ब्लॉक नंबर = ब्लॉक j module 2^C

आकृती 5.6 थेट मॅपिंग तंत्र दर्शविते ज्यामध्ये कॅशेमध्ये 128 ब्लॉक असतात आणि प्रत्येक कॅशे ब्लॉकचा साईझ 16 (2^4) वर्ड असतो. मुख्य मेमोरी ऍड्रेस $12+4 = 16$ बिट्स आहे, जेथे मेमोरी ऍड्रेसचे सर्वात कमी लक्षणीय चार बिट्स वर्ड ऍड्रेस दर्शवतात. कॅशे ब्लॉक ऍड्रेससाठी आवश्यक बिट्सची संख्या 128 आहे (2^7). टॅग ऍड्रेससाठी आवश्यक असलेल्या बिट्सची एकूण संख्या खालीलप्रमाणे फिक्स्ड केली जाते:

टॅग ऍड्रेससाठी आवश्यक बिट्सची एकूण संख्या = मुख्य मेमोरी ऍड्रेसतील बिट्सची संख्या - ब्लॉक ऍड्रेससाठी बिट्सची संख्या - वर्ड ऍड्रेससाठी बिट्सची संख्या

म्हणून, टॅग ऍड्रेससाठी आवश्यक एकूण बिट्सची संख्या = $16-7-4 = 5$ बिट्स.

कॅशे ब्लॉक 0 मध्ये मुख्य मेमोरी तील ब्लॉक 0,128,256,..., आणि कॅशे ब्लॉक 1 मध्ये मुख्य मेमोरी तील ब्लॉक 1,129,257,... ब्लॉक मॅपिंग केला जातात.

उदाहरणार्थ, मुख्य मेमोरी ब्लॉक 856 कॅशे ब्लॉक 856 मॉड्यूलो 128 = 88 मध्ये मॅपिंग केला जाईल.

उदाहरण 5.1

तुमच्या संगणकाची बाइट-ॲड्रेसेबल मेमोरी 2^{32} बाइट आकाराची आहे आणि ती ब्लॉकमध्ये विभागलेली आहे याचा विचार करा. डेटाच्या एका ब्लॉकमध्ये 32 बाइट असू शकतात. या संगणकात 512-ब्लॉकचा थेट मॅप केलेला कॅशे आहे. कॅशे मेमोरीच्या टॅग फील्डद्वारे किती बिट्स वापरले जातात?

उपाय:

ही एक बाइट संबोधित करण्यायोग्य मेमोरी आहे आणि मुख्य मेमोरी आकाराने 2^{32} बाइट आहे. तर मुख्य मेमोरीचे भौतिक संबोधित करण्यायोग्य 32 बिट्स लांब आहे.

प्रत्येक ब्लॉकचा साईझ $32 = 2^5$ बाइट आहे. 32 बिट मुख्य मेमोरी ऍड्रेसचे पहिले 5 बिट वर्ड ऍड्रेस दर्शवतात.

कॅशे मेमोरीसाठी $512 = 2^9$ कॅशे ब्लॉक आहेत आणि ते थेट माप केलेले कॅशे असल्यामुळे, 9 बिट्स कॅशे ब्लॉक पत्त्यांसाठी वापरले जातात.

मुख्य मेमोरी भौतिक ऍड्रेस 32 बिट्स



डॅरेक्ट मॅपिंग केलेल्या कॅशेसाठी मुख्य मेमोरी भौतिक ऍड्रेसच्या बिट्सची संख्या कॅशे टॅग ऍड्रेसमध्ये वापरल्या जाणार्या बिट्सची संख्या + ब्लॉक ऍड्रेससाठी बिट्सची संख्या + वर्ड ऍड्रेससाठी बिट्सची संख्या यांच्या बेरीज इतकी असते.

अशाप्रकारे, (कॅशेच्या टॅग ऍड्रेसमध्ये वापरलेल्या बिट्सची संख्या) $+(9)+(5) = 32$.

कॅशे टॅग ऍड्रेसमध्ये वापरलेल्या बिट्सची संख्या $= 32-14 = 18$.

थेट मॅपिंग तंत्राचे खालील फायदे आणि तोटे आहेत

फायदा:

- 1) डॅरेक्ट मॅपिंग चा दृष्टीकोन सोपा आणि अंमलात आणणे सोपे आहे.
- 2) वर्ड शोध जलद असतात कारण केवळ टॅग फील्ड जुळले पाहिजे.
- 3) टॅग फील्ड लहान आहे.
- 4) असोसिएटिव्ह आणि सेट-असोसिएटिव्ह कॅशे मॅपिंग पेक्षा डॅरेक्ट मॅपिंग कमी खर्चिक आहे.

तोटा:

संघर्ष चुकल्यामुळे थेट मॅपिंग ची कामगिरी वाईट होते. रिकामे कॅशे ब्लॉक उपलब्ध असले तरी, भरलेले ब्लॉक वारंवार बदलले जातात कारण नवीन ब्लॉक कॅशेमधील फिक्स्ड स्थानावर मॅप केला जाऊ शकतो. रिकामे कॅशे ब्लॉक्स असूनही भरलेले कॅशे ब्लॉक्स बदलले जातात तेव्हा कॉन्फ्लिक्ट मिस होते.

2. असोसिएटिव्ह मॅपिंग

असोसिएटिव्ह मॅपिंग हा सर्वात कार्यक्षम आणि लवचिक मॅपिंग दृष्टीकोन आहे. हे टॅग ऍड्रेस आणि वर्ड ऍड्रेस वापरून ब्लॉक ओळखते. असोसिएटिव्ह मॅपिंग मुळे, कॅशेमधील प्रत्येक स्थान मुख्य मेमोरीतून कोणताही वर्ड संचयित करू शकते.

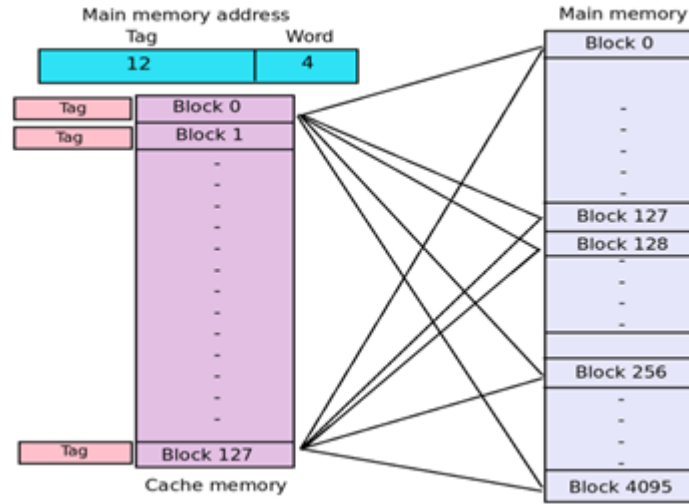
आकृती 5.7 मध्ये विशिष्ट असोसिएटिव्ह मॅपिंग तंत्र दर्शविले आहे. समजा मुख्य मेमोरीची क्षमता 4096 (2^{12}) ब्लॉक्सची आहे. प्रत्येक ब्लॉक आकारात 16 (2^4) बाइट आहे. मुख्य मेमोरीमध्ये भौतिक ऍड्रेस $12+4 = 16$ बिट्स आहे. कारण कॅशे मेमोरीमध्ये 128 ब्लॉक असू शकतात आणि वर्ड ऍड्रेससाठी आवश्यक बिट्सची संख्या 4 आहे. तर, कॅशे टॅग ऍड्रेस दर्शविण्यासाठी आवश्यक बिट्सची संख्या $16-4 = 12$ बिट्स आहे.

जेव्हा कॅशे भरलेला असतो, तेव्हा कॅशेमध्ये नवीन ब्लॉक आणण्यासाठी विद्यमान ब्लॉक बदलला जातो. या प्रकरणात, बदलण्यासाठी ब्लॉक निवडण्यासाठी एक अल्गोरिदम वापरला जातो. अनेक रिप्लेसमेंट अल्गोरिदम शक्य आहेत, परंतु कमीतकमी अलीकडे वापरलेले (लीस्ट रेसेन्टली यूज्ड एलआरयू) असोसिएटिव्ह कॅशेसाठी लोकप्रियपणे वापरले जातात.



स्कॅन करा

एलआरयू प्रतिस्थापन
अल्गोरिदम



आकृती. 5.7: असोसिएटिव्ह मॅपिंग

उदाहरण 5.2

256 बाइट ब्लॉक आकाराच्या 16 KB सहयोगी मॅप केलेल्या कॅशेचा विचार करा. मुख्य मेमोरी साईझ 128 KB आहे. टॅगच्या ऍड्रेससाठी किती बिट्स वापरले जातात?

उत्तर:

कॅशे मेमोरी साईझ = 16KB हे दिले आहे

ब्लॉक साईझ-256 बाइट

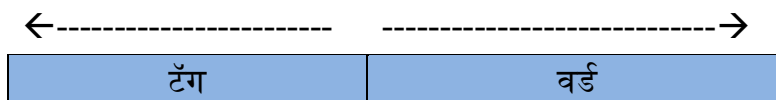
मुख्य मेमोरी साईझ = 128 KB

मेमोरी ही बाइट संबोधित करण्यायोग्य मानली जाते.

मुख्य मेमोरी कॅपॅसिटी = 128 Kb = $2^7 \times 2^{10} = 2^{17}$ बिट्स

अशा प्रकारे मुख्य मेमोरी भौतिक ऍड्रेसतील बिट्सची संख्या = 17 बिट्स

मुख्य मेमोरी भौतिक ऍड्रेस 17 बिट्स

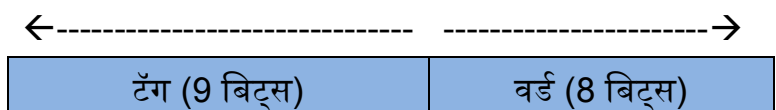


कॅशे मेमोरीचा ब्लॉक साईझ 256 बाइट = 2^8 बिट

वर्ड ऍड्रेससाठी बिट्सची संख्या = 8 बिट्स

अशा प्रकारे, मुख्य मेमोरी ऍड्रेसचे पहिले 8 बिट्स वर्ड पत्त्यांसाठी वापरले जातात.

मुख्य मेमोरी भौतिक ऍड्रेस 17 बिट्स



म्हणून, टॅग ऍड्रेससाठी बिट्सची संख्या = भौतिक ऍड्रेसतील बिट्सची संख्या - वर्ड ऍड्रेससाठी बिट्सची संख्या =

17 बिट्स- 8 बिट्स = 9 बिट्स

अशा प्रकारे, टॅग एंड्रेससाठी बिटची संख्या = 9 बिट

असोसिएटिव्ह मॅपिंग चे खालील फायदे आणि तोटे आहेत

फायदा:

1) जेव्हा कॅशमध्ये नवीन ब्लॉक वाचला जातो तेव्हा असोसिएटिव्ह मॅपिंग हा सर्वात लवचिक दृष्टीकोन असतो.

2) इतर दोन कॅशे मॅपिंग तंत्रांपेक्षा हिट रेट चांगला आहे.

तोटा:

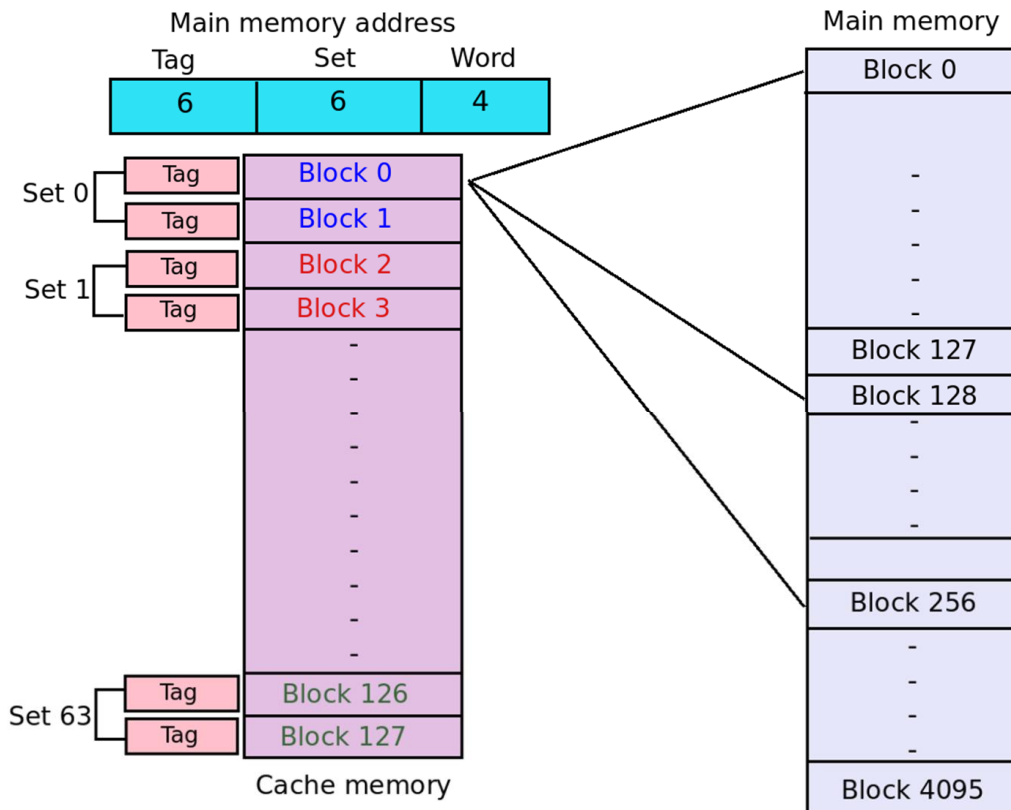
1) इच्छित कॅशे ब्लॉक फिक्स्ड करण्यासाठी सर्व टॅग नमुने शोधणे आवश्यक आहे. त्यामुळे असोसिएटिव्ह कॅशे हे डायरेक्ट-मॅप केलेल्या कॅशेपेक्षा अधिक क्लिष्ट असतात.

2) दीर्घ शोध विलंब टाळण्यासाठी टॅग समांतर शोधले जातात. परंतु अशा प्रकारचा शोध अंमलात आणणे महागडे आहे.

3) टॅग फील्ड लांब आहे.

3. सेट- सोसिएटिव्ह मॅपिंग

सेट-असोसिएटिव्ह मॅपिंगमध्ये डायरेक्ट आणि असोसिएटिव्ह मॅपिंग या दोन्हीची वैशिष्ट्ये एकत्रित केली जातात. संचातील कोणत्याही गटामध्ये मुख्य मेमोरी ब्लॉक समाविष्ट केला जाऊ शकतो. हे मॅपिंग डायरेक्ट मॅपिंगपेक्षा अधिक लवचिकता प्रदान करते, तथापि ते असोसिएटिव्ह मॅपिंग इतके लवचिक नाही.



आकृती. 5.8: सेट-असोसिएटिव्ह मॅपिंग

आकृती 5.8 मध्ये 64 संचांमध्ये आयोजित 128 कॅशे ब्लॉक्ससह सेट-असोसिएटिव्ह कॅशे मेमोरी दर्शविली आहे, प्रत्येक संचामध्ये दोन कॅशे ब्लॉक्स आहेत. संचांची एकूण संख्या 64 (2^6) आहे हे लक्षात घेता, 6 बिट्सचा समावेश असलेल्या ऍड्रेसचा वापर करून एक विशिष्ट सेट ओळखला जाऊ शकतो. जर सहा बिट पैकी प्रत्येक बिट 0 असेल तर तो पहिला सेट दर्शवितो. प्रत्येक ब्लॉकमध्ये 16 बिट्स असतात; म्हणून, ब्लॉकमधील विशिष्ट वर्ड ओळखण्यासाठी 4-बिट बायनरी व्हॅल्यू आवश्यक असते. जर चार सर्वात कमी लक्षणीय बिट्स 0000 असतील, तर पहिला वर्ड ब्लॉक ओळखला जातो.

चार बिट्स 0011 आहेत, जे सूचित करतात की हा ब्लॉकमधील चौथा वर्ड आहे. कारण प्रत्येक ब्लॉकमध्ये 4096 ब्लॉक आणि 16 बिट्स आहेत, मुख्य मेमोरीचा भौतिक ऍड्रेस 16 बिट्स लांब आहे आणि मेमोरीची एकूण क्षमता $2^{12+4} = 2^{16}$ आहे.

उदाहरण 5.3

ब्लॉक आकाराच्या 256 बाइट्सह 16 KB आकाराच्या 2-वे सेट असोसिएटिव्ह मॅप केलेल्या कॅशेचा विचार करा. मुख्य मेमोरी साईझ 128 KB आहे. टॅग, सेट आणि वर्ड ऍड्रेससाठी आवश्यक असलेल्या बिट्सची संख्या मोजा.

उपाय: हे लक्षात घेता

सेट साईझ = 2 फिक्स्ड करा

कॅशे मेमोरी साईझ = 16 KB

ब्लॉक साईझ = 256 बाइट

मुख्य मेमोरी साईझ = 128 KB

मेमोरी ही बाइट संबोधित करण्यायोग्य मानली जाते.

मुख्य मेमोरीचा साईझ = 128 KB = 2.7 बाइट

अशाप्रकारे, भौतिक ऍड्रेसतील बिट्सची संख्या = 17 बिट

कॅशेमधील ब्लॉक्सची एकूण संख्या = कॅशे साईझ/ब्लॉक साईझ = 16 KB/256 बाइट = 2.14 बाइट/2.8 बाइट = 2.6 ब्लॉक

अशाप्रकारे, कॅशेमधील ब्लॉकची संख्या = 64 ब्लॉक

कॅशेमधील संचांची एकूण संख्या = कॅशेमधील ब्लॉकची एकूण संख्या/सेट साईझ = 64/2 = 32 सेट = 2.5 सेट

अशा प्रकारे, सेट क्रमांकातील बिट्सची संख्या = 5 बिट्स

मुख्य मेमोरी भौतिक ऍड्रेस 17 बिट्स

←----->

टॅग (4 बिट्स)	सेट(5 बिट्स)	वर्ड (8 बिट्स)
---------------	--------------	----------------

टॅगमधील बिट्सची संख्या = भौतिक ऍड्रेसतील बिट्सची संख्या - (संचातील बिट्सची संख्या + वर्ड ऍड्रेसतील बिट्सची संख्या) = 17 बिट्स - (5 बिट्स + 8 बिट्स) = 17 बिट्स - 13 बिट्स = 4 बिट्स

अशाप्रकारे टॅगमधील बिट्सची संख्या = 4 बिट्स

कॅशे सेट 0 माप्स मेमोरी ब्लॉक्स 0,64,128,..., 4032 त्याच्या दोन ब्लॉक ऍड्रेसवर. आवश्यक ब्लॉक अस्तित्वात आहे की नाही हे निर्धारित करण्यासाठी, ऍड्रेस टॅग क्षेत्राची तुलना संचातील दोन ब्लॉक्सच्या टॅगशी केली जाणे आवश्यक आहे. ही द्विमार्गी असोसिएटिव्ह शोध अंमलबजावणी सोपी आहे. प्रत्येक संचातील ब्लॉकची संख्या प्रणालीच्या आवश्यकतेनुसार सुधारित केली जाऊ शकते.

सेट-असोसिएटिव्ह मॅपिंग चे खालील फायदे आणि तोटे आहेत:

फायदा:

- 1) अनेक ब्लॉक पोजिशनिंग पर्याय असल्यामुळे थेट मापिंगच्या वादाची समस्या कमी होते.
- 2) असोसिएटिव्ह शोधांची संख्या कमी करून, हार्डवेअर खर्च कमी केला जाऊ शकतो.

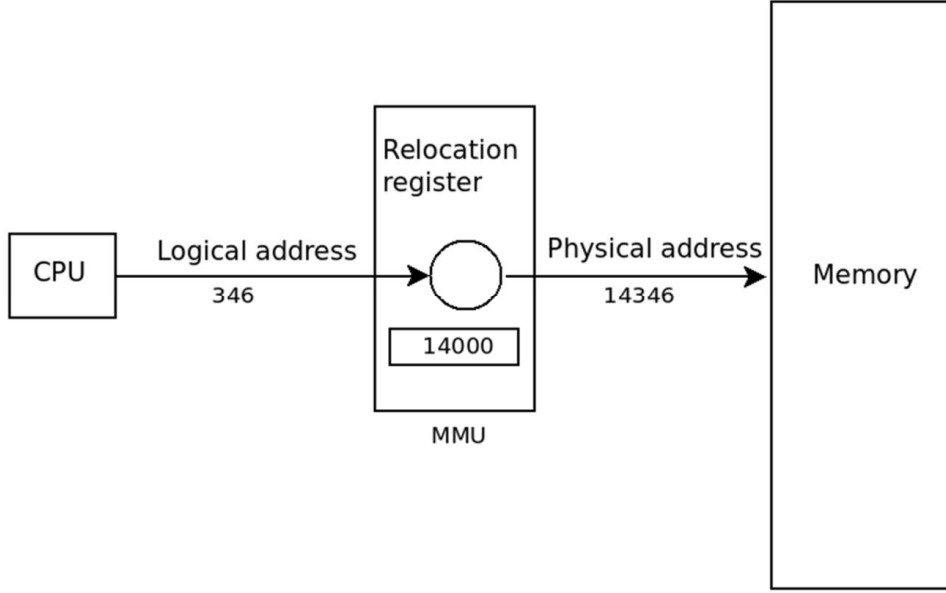
तोटा:

- 1) सेट-असोसिएटिव्ह कॅशे मेमोरी खूप महाग आहे. सेटच्या आकारानुसार खर्च वाढतो.

एकंदरीत, असोसिएटिव्ह मॅपिंग इतर सर्वापेक्षा चांगली कामगिरी करते, परंतु त्याची अंमलबजावणी महागडी आहे. म्हणून, सामान्य पद्धतींमध्ये सेट-संबंधित मापिंगला प्राधान्य दिले जाते [4].

मेमोरी व्यवस्थापन एकक (MMU)

प्रोसेसर-जनरेट केलेला लॉजिक ऍड्रेस आणि मुख्य मेमोरीचा भौतिक ऍड्रेस यांच्यातील रन-टाइम मॅपिंग मेमोरी व्यवस्थापन युनिट म्हणून ओळखल्या जाणार्या हार्डवेअर घटकाद्वारे केले जाते. (MMU). लॉजिकल ऍड्रेस हा प्रोग्राम चालू असताना CPU द्वारे व्युत्पन्न केलेला आभासी ऍड्रेस असतो. जेव्हा प्रक्रिया डिस्क आणि मुख्य मेमोरी दरम्यान हलतात, तेव्हा ऑपरेटिंग सिस्टम ते कसे आत आणि बाहेर हलवतात हे व्यवस्थापित करते. मेमोरीची उपलब्धता आणि वापर या दोन्हीवर कार्यप्रणालीद्वारे लक्ष ठेवले जाते.



आकृती. 5.9: मेमोरी व्यवस्थापन युनिटचा वापर करून भौतिक ऍड्रेसच्या मापिंगसाठी लॉजिक ऍड्रेस

CPU द्वारे तयार केलेला ऍड्रेस हा लॉजिक ऍड्रेस असतो. मेमोरी व्यवस्थापन एकक (एमएमयू) ऍड्रेसतर रजिस्टरच्या ब व्हॅल्यूत लॉजिक ऍड्रेस एल जोडून वास्तविक मुख्य मेमोरी ऍड्रेस (भौतिक ऍड्रेस) तयार करते. (base register). प्रोसेसर या ठिकाणाहून प्रोग्रामच्या अंमलबजावणीसाठी डेटा किंवा इंस्ट्रक्शन मिळवू शकतो. आकृती 5.9 दर्शविते की मुख्य मेमोरीचा भौतिक ऍड्रेस 14346 तयार करण्यासाठी सीपीयूचा लॉजिक ऍड्रेस 346 कसे रीलोकेशन रजिस्टर व्हॅल्यू 14000 मध्ये जोडला जातो जिथे ही इंस्ट्रक्शन/डेटा संग्रहित केला जाईल.

उदाहरण 5.4

CPU ने तयार केलेला ऍड्रेस 500 आहे आणि ऍड्रेसतर रजिस्टर व्हॅल्यू 10000 आहेत. मग डेटामध्ये प्रवेश करण्यासाठी मेमोरीचा भौतिक ऍड्रेस काय आहे?

उत्तर: हे लक्षात घेता की लॉजिकल ऍड्रेस 500 हे रीलोकेशन रजिस्टर (बेस रजिस्टर) व्हॅल्यू 10000 सह जोडले गेले आहे आणि रिझल्ट 10500 MMU पासून मेमोरीमध्ये तयार केले गेले आहे. हा परिणाम 10500 हा या स्थानावरील प्रोसेसरचा वास्तविक मेमोरी ऍड्रेस (भौतिक ऍड्रेस) असेल जो प्रोग्रामच्या अंमलबजावणीसाठी डेटा किंवा इंस्ट्रक्शनमध्ये प्रवेश करेल.

5.2.4 रीड ओन्ली मेमोरी (ROM)

रीड ओन्ली मेमोरी (रॉम) हा प्राथमिक मेमोरीचा एक प्रकार आहे परंतु ही मेमोरी प्रणाली प्रोग्राम धारण करते. याला सिस्टम मेमोरी म्हणून संबोधले जाते आणि प्रोसेसर केवळ रॉम चिपमधून डेटा वाचू शकतो. ही एक अस्थिर नसलेली मेमोरी आहे आणि शक्ती काढून टाकल्यानंतरही डेटा कायमस्वरूपी संग्रहित केला जातो [5].

रॉम चिपच्या संरचनेत एन-बिट ऍड्रेस आणि डी-बिट युनिटायरेक्शनल डेटा बस असतात. या रॉमची साठवण क्षमता $2^n \times d$ म्हणून व्यक्त केली जाऊ शकते. 9-बिट ऍड्रेस

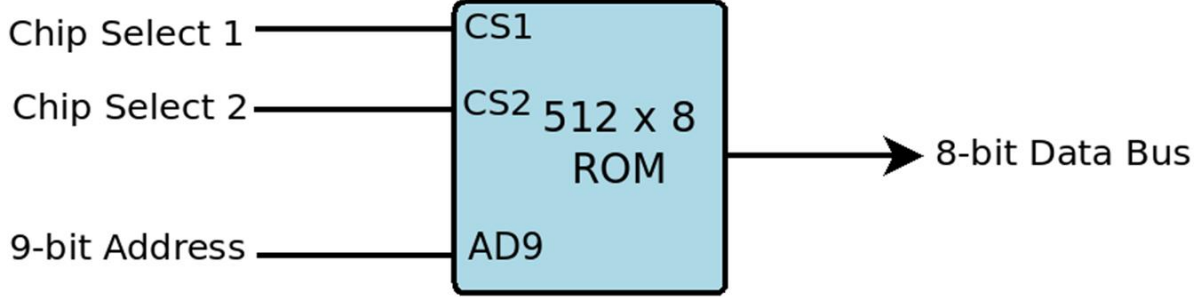


स्कॅन करा

RAM विरुद्ध ROM
मेमोरी

लाईन्स आणि 8-बिट डेटा लाईन्स असलेली रॉम चिप आकृती 5.10 मध्ये दर्शविली आहे. 9 बिट ऍड्रेससह, $2^9 = 512$ मेमोरी स्थाने आहेत आणि प्रत्येक मेमोरी स्थान 8 बिट सेटयित करू शकते. कालांतराने, विविध अनुप्रयोगांमध्ये वापरण्यासाठी रॉमच्या विविध आवृत्त्या विकसित झाल्या आहेत.

- रॉम ही एक अस्थिर नसलेली मेमोरी आहे ज्यामध्ये डेटा एकदा लिहिला जातो आणि कधीही बदलला जात नाही. मूलभूत इनपुट/आउटपुट प्रणाली (BIOS) नावाचा फर्मवेअर प्रोग्राम रॉममध्ये संग्रहित केला जातो.



आकृती. 5.10: 9-ऍड्रेस लाईन्स आणि 8 युनिडायरेक्शनल डेटा लाईन्ससह एक टिपिकल रॉम चिप

- PROM हे 'प्रोग्रामेबल रीड-ओनली मेमोरी' चे संक्षिप्त रूप आहे. या प्रकारची मेमोरी केवळ एकदाच प्रोग्राम केली जाऊ शकते आणि ती प्रोग्राम करण्यासाठी "PROM प्रोग्रामर" म्हणून ओळखल्या जाणार्या विशेष साधनाची आवश्यकता असते.
- R PROM चा अर्थ 'इलेक्ट्रिकली प्रोग्रामेबल रीड-ओनली मेमोरी' असा आहे. मेमोरी प्रोग्राम केल्यानंतर, ती मिटवण्यासाठी अतिनील प्रकाशाचा वापर केला जाऊ शकतो.
- EEPROM म्हणजे 'इलेक्ट्रिकली इरेझेबल प्रोग्रामेबल रीड-ओनली मेमोरी'. विद्युत विभाजनासह, तुम्ही त्यावर माहिती लिहू शकता आणि ती मिटवू शकता.
- फ्लॅश मेमोरी तुम्हाला डेटा लिहिण्याची आणि मिटवण्याची परवानगी देते. डेटा पुन्हा लिहिला जाऊ शकतो. डिजिटल कॅमेरे आणि सेल फोन दोन्ही फ्लॅश मेमोरी कार्ड वापरतात.

उदाहरण 5.5

2048 x 16 रॅम आणि रॉम चिप्ससाठी, किती ऍड्रेस लाइन आणि डेटा लाइन आहेत?

उत्तर: याची चिप कॅपेसिटी 2048 x 16 पिक्सल आहे. $2048 = 2^{11}$ पासून, रॅम चिपमध्ये 11 ऍड्रेस लाईन्स आणि 16 बायडायरेक्शनल डेटा लाईन्स आहेत आणि रॉम चिपमध्ये 11 ऍड्रेस लाईन्स आणि 16 युनिडायरेक्शनल डेटा लाईन्स आहेत.

5.3 सेकंडरी मेमोरी

सेकंडरी मेमोरी ही क्रमिक प्रवेश मेमोरी तंत्रज्ञानावर आधारित आहे. डेटा संलग्न मेमोरी ब्लॉक्समध्ये वाटप केला जाऊ शकतो आणि डेटा संलग्न मेमोरी ब्लॉक्समधून हटविला जाऊ शकतो. ही अस्थिर नसलेली मेमोरी आहे, डेटा कायमस्वरूपी संग्रहित केला जाऊ शकतो. माग्नेटिक डिस्क, माग्नेटिक टेप, CD-



स्कॅन करा
दुय्यम मेमोरीचे प्रकार
जाणून घेण्यासाठी

ड्राइव्ह इ. हे कायमस्वरूपी साठवणुकीच्या उपकरणांचे उदाहरण आहे. इतर सर्व स्मृतींच्या तुलनेत या उपकरणांमध्ये साठवण क्षमता जास्त असते. साठवण क्षमता GB ते TB पर्यंत आहे. CPU प्रवेश वेळ इतर सर्व स्मृतींपेक्षा जास्त आहे.

हार्ड डिस्कमध्ये, डेटा ट्रॅक नावाच्या रिंगच्या एकाग्र संचामध्ये आयोजित केला जातो. प्लॅटर तील काही भाग वाचण्या/लिहिण्या साठी वाचन/लेखन शीर्ष वापरले जाते. प्रत्येक मार्ग पुढे विभागांमध्ये विभागलेला आहे. प्रत्येक क्षेत्राचा सार्वत्रिक साईझ 512 बाइट आहे. डेटाची घनता सर्वात आतील मार्गावर जास्त असते कारण आतील भागात मेमोरीसाठी कमी जागा असते. सर्वात बाहेरील ट्रॅकमध्ये डेटाची घनता कमी असते कारण त्याची फील्ड मेमोरी क्षमता मोठी असते. बाह्य मार्गांमध्ये मेमोरीची जागा वाया जाते. मेमोरीचा अपव्यय रोखण्यासाठी फील्डऐवजी फील्डचा वापर केला जातो. प्रत्येक ट्रॅकवर समान लांबीची फील्ड आहेत जिथे फिक्स्ड प्रमाणात डेटा ठेवला जातो. बाह्य मार्गाच्या तुलनेत, ज्यामध्ये अधिक फील्ड आहेत, आतील मार्गावर कमी फील्ड आहेत. डिस्कची क्षमता अशी मोजली जाऊ शकते.

डिस्क क्षमता = पृष्ठभाग x ट्रॅक x फील्ड x बाइट

डिस्क क्षमता ही #surfaces, #tracks प्रति पृष्ठभाग, #sectors प्रति ट्रॅक आणि #bytes प्रति सेक्टर यांचे गुणाकार आहे. येथे #हे चिन्ह दिलेल्या एककांची संख्या दर्शवते.

डिस्क कामगिरी: डिस्कची कार्यक्षमता सीक टाईम, रोटेशन लेटन्सी आणि डेटा ट्रान्सफर रेटवर अवलंबून असते.

- सीक टाईम (T_S) वाचन/लेखन शीर्षासाठी योग्य ट्रॅक शोधण्यासाठी लागणारा वेळ.
- रोटेशनल लेटेंसी (T_R) डिस्क नियंत्रक रोटेशनल लेटेंसी म्हणून ओळखल्या जाणार्या निवडलेल्या ट्रॅकच्या शीर्षस्थानी रांगेत उभे राहण्यासाठी इच्छित सेक्टर स्पिन होईपर्यंत प्रतीक्षा करतो (T_R). पूर्वनिर्धारित रोटेशनल विलंबता $T_R = (1/2) \times$ रोटेशन वेळ आहे.

- डिस्क प्रवेश वेळ (T) डिस्क प्रवेश वेळ शोध वेळ आणि रोटेशनल विलंबता यांची बेरीज आहे.

$$T = T_S + T_R$$

- ट्रान्सफर वेळ (T_T) डिस्क रोटेशनल गती (r) ट्रान्सफर वेळ खालीलप्रमाणे ठरवते

$$T_T = (b/r) \times N$$

प्रसारित केल्या जाणाऱ्या बाइटची संख्या (b) रोटेशनल स्पीड (r) ने प्रति सेकंद क्रांतीमध्ये विभागली जाते, जी नंतर ट्रॅकवरील बाइटची संख्या (N) ने गुणाकार केली जाते.

$$\text{डिस्कचा सरासरी प्रवेश वेळ (Tavg)} = T_S + T_R + T_T$$

$$T_{avg} = T_S + (1/2) \times r + (b/r) \times N$$

उदाहरण 5.6

डिस्कमध्ये 8 डेटा रेकॉर्डिंग पृष्ठभाग असतात.

प्रत्येक पृष्ठभागावर 4096 मार्गिका आहेत.

प्रत्येक मार्गावर 256 विभाग आहेत

प्रत्येक क्षेत्रात 512 बाइट असतात.

डिस्कची एकूण क्षमता किती आहे?

उत्तर: डिस्क क्षमता = #surface x #ट्रॅक x #sectors x #bytes

एकूण डिस्क क्षमता = 8 x 4096 x 256 x 512 = 4294967296 बाइट

5.4 प्रोग्रामेबल पेरिफेरल इंटरफेस

मायक्रोप्रोसेसरच्या इनपुट/आउटपुट इंटरफेसची क्षमता वाढविण्यासाठी 8255 IC. मध्ये 24 इनपुट/आउटपुट पिन आहेत. सर्व आय/ओ पिन A, B आणि C पोर्टमध्ये वर्गीकृत केल्या आहेत. A आणि B हे दोन्ही पोर्ट 8-बिट इनपुट/आउटपुट पोर्ट म्हणून काम करतात. पोर्ट C हे 8-बिट आय/ओ पोर्ट, दोन 4-बिट आय/ओ पोर्ट किंवा A आणि B पोर्टसाठी हँडशेक पोर्ट म्हणून वेगवेगळ्या प्रकारे संरचित केले जाऊ शकते. ही पोर्ट दोन गटांमध्ये विभागली गेली आहेत.

- गट अ: पोर्ट A आणि पोर्ट C वरचा भाग
- गट ब: पोर्ट B आणि पोर्ट C खालचा भाग

एँड्रेस लाइन A1 आणि A0 च्या व्हॅलूनुसार, प्रत्येक पोर्ट किंवा कंट्रोल रजिस्टर टेबल 5.2 मध्ये सारांशित केल्याप्रमाणे डेटा रजिस्टरमध्ये प्रवेश करतो.

तक्ता 5.2: पोर्ट निवडीसाठी एँड्रेसच्या लाइनची बिट व्हॅलू

A1	A0	निवडलेले पोर्ट
0	0	पोर्ट A
0	1	पोर्ट B
1	0	पोर्ट C
1	1	कंट्रोल रजिस्टर

5.4.1 ऑपरेशनल मोड 8255

8-बिट कंट्रोल रजिस्टर, तक्ता 5.2 मध्ये दर्शविल्याप्रमाणे, केवळ ऑपरेशन मोड नियंत्रित करत नाही तर इनपुट/आउटपुट साठी पोर्ट देखील ओळखते. बिट सेट/रीसेट मोड आणि इनपुट/आउटपुट मोड हे दोन ऑपरेशनल मोड आहेत. हे मोड कंट्रोल रजिस्टरच्या सर्वात महत्त्वपूर्ण बिट (MSB) D7 द्वारे निर्धारित केले जातात.

तक्ता 5.3: बीएसआर पद्धतीसाठी कंट्रोल रजिस्टर स्वरूप

D7	D6	D5	D4	D3	D2	D1	D0
0	X	X	X	B2	B1	B0	S/R

↓ Always 0 for BSR mode ↓ Don't care ↓ Port C selection bits ↓ Set/Reset

• बिट सेट/रीसेट (BSR) पद्धती

जर D7 बिट 0 वर सेट केले असेल, तर 8255 हे BSR मोडमध्ये चालते, जे फक्त पोर्ट C वर उपलब्ध आहे. कंट्रोल रजिस्टर व्हॅल्यू बदलून, PC0 ते PC7 पर्यंत पोर्ट सीची प्रत्येक ओळ सेट/रीसेट केली जाऊ शकते. जरी BSR आणि आय/ओ पद्धती स्वतंत्रपणे अस्तित्वात असल्या, तरी कोणत्याही पद्धतीचा दुसऱ्याच्या कार्यपद्धतीवर परिणाम होत नाही.

उर्वरित बिट्स D6, D5 आणि D4 आहेत काळजी करू नका (X). पोर्ट C पिन निवडण्यासाठी D3, D2 आणि D1 हे बिट्स वापरले जातात. पोर्ट C पिन सेट/रीसेट करण्यासाठी बिट D0 चा वापर केला जातो.

• इनपुट/आउटपुट मोड (I/O मोड)

जेव्हा कंट्रोल रजिस्टरचा D7 बिट 1 वर सेट केला जातो तेव्हा हा I/O मोड निवडला जातो. यात तीन भिन्न आय/ओ मोड आहेत: मोड 0, मोड 1 आणि मोड 2.

1. मोड 0: मोड 0 मध्ये, पोर्ट A आणि B हातमिळवणी न करता मूलभूत आय/ओ ऑपरेशन्स करू शकतात. पोर्ट C हे एकच 8-बिट पोर्ट किंवा दोन 4-बिट पोर्ट असू शकतात. पोर्ट C चे दोन भाग इनपुट आणि आउटपुट पोर्ट म्हणून सेट केले जाऊ शकतात कारण ते वेगळे आहेत.

तक्ता 5.4: पोर्ट C ची पिन निवड

D3	D2	D1	पोर्ट C ची पिन निवड
0	0	0	PC0
1	0	1	PC1
0	1	0	PC2
1	1	1	PC3
0	0	0	PC4
1	0	1	PC5
0	1	0	PC6
1	1	1	PC7

2. मोड 1: मोड 1 वापरताना वेगवेगळ्या मोडमध्ये कार्य करण्यासाठी पोर्ट A आणि B सेट करणे शक्य आहे. उदाहरणार्थ, पोर्ट A मोड 0 मध्ये चालू शकतो तर पोर्ट B मोड 1 मध्ये चालतो. याव्यतिरिक्त, हस्तांदोलन इनपुट किंवा आउटपुट एकतर पोर्ट A किंवा पोर्ट B वर पाठवले जाऊ शकते. तुम्ही कोणतेही पोर्ट वापरू शकता. पोर्ट सीच्या पिनच्या एका भागावर हस्तांदोलन लाइन लागू केल्या जातात.

3. मोड 2: मोड 2 मध्ये, PA0 ते PA7 पिनद्वारे द्विदिशात्मक हस्तांदोलन डेटा ट्रान्सफरसाठी (त्याच आठ ओळी इनपुट किंवा आउटपुटसाठी वापरल्या जाऊ शकतात) केवळ पोर्ट A सुरू केला जाऊ शकतो. PC3 ते PC7 या पोर्ट A च्या हस्तांदोलन लाइन आहेत. जर गट B मोड 0 मध्ये प्रारंभ केला असेल तर पोर्ट C पिन PC0-PC2 इनपुट/आउटपुट लाइन म्हणून वापरले जाऊ शकते. गट B उर्वरित पिन पोर्ट B हस्तांदोलन करण्यासाठी मोड 1 मध्ये वापरू शकतो.

5.4.2 प्रोसेसरशी जोडणी

एक्सटर्नल उपकरणांसह प्रोसेसरच्या परस्परसंवादासाठी खालील कार्ये केली जातात:

1. प्रोसेसर/सीपीयूकडून मिळालेल्या आदेशानुसार आय/ओ मॉड्यूल कंट्रोल बसवर सिग्नल पाठवते.
2. CPU आणि I/O मॉड्यूल डेटाची देवाणघेवाण करतात.
3. आय/ओ मॉड्यूल BUSY आणि READY सारख्या स्थिती संकेतांद्वारे CPU ला त्याची स्थिती कळवते.
4. आय/ओ मॉड्यूल प्रत्येक एक्सटर्नल उपकरणाला एका विशिष्ट ऍड्रेससह ओळखते.
5. आय/ओ मॉड्यूल एक्सटर्नल उपकरणांकडून स्थिती संकेत प्राप्त करते. डिव्हाइसची स्थिती आय/ओ मॉड्यूलद्वारे CPU कडे पाठवली जाते.
6. आय/ओ मॉड्यूलचा वापर करून डेटा पाठवला आणि प्राप्त केला जातो.
7. आय/ओ मॉड्यूल डेटा तात्पुरता बफरमध्ये ठेवतो जे मुख्य मेमोरीमधून जलद स्फोटात येत आहेत. हा डेटा एक्सटर्नल उपकरणाला त्याच्या डेटा दरानुसार पाठवला जातो.
8. आय/ओ मॉड्यूल CPU ला आढळलेल्या कोणत्याही त्रुटींचा अहवाल देखील देते. प्रिंटआउट्स घेताना, जर प्रिंटआऊट मशीनमध्ये कागद जाम झाला असेल तर आय/ओ मॉड्यूल पेपर जाम त्रुटी शोधते आणि प्रोसेसरला अहवाल देते. पॅरिटी चेकर्स वापरून डेटा ट्रान्समिशन त्रुटी आढळली.

एक्सटर्नल उपकरणांसह प्रोसेसरचे इंटरफेस खालील तीन पद्धतींपैकी कोणत्याही एका पद्धतीने केले जाऊ शकते:

1. मेमोरी/(I/O) माप I/O

जेव्हा आय/ओ डिव्हाइस मेमोरी मध्ये माप केले जाते, तेव्हा मेमोरी आणि आय/ओ डिव्हाइस या दोघांद्वारे वापरलेली ऍड्रेस स्पेस समान असते. या विशिष्ट प्रकारच्या आय/ओ ला 'मेमोरी-माप आय/ओ' म्हणतात. जेव्हा आय/ओ उपकरण आय/ओ जागेत माप केले जाते, तेव्हा आय/ओ उपकरणासाठीची ऍड्रेसची जागा मेमोरी उपकरणासाठीच्या ऍड्रेसच्या जागेपेक्षा वेगळी असते. या पद्धतीला 'आय/ओ-माप आय/ओ' म्हणतात.

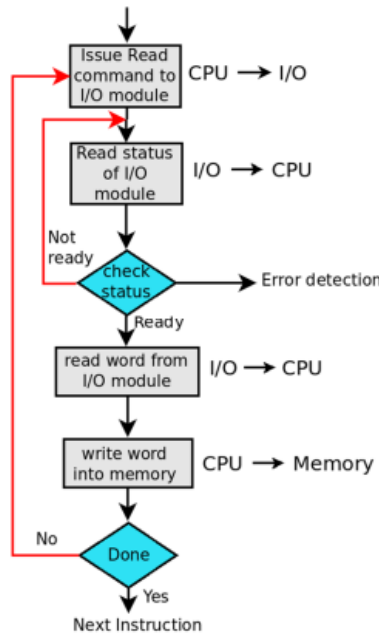


स्कॅन करा
मॅप केलेल्या I/O विरुद्ध
I/O मॅप केलेले I/O
मेमोरीसाठी

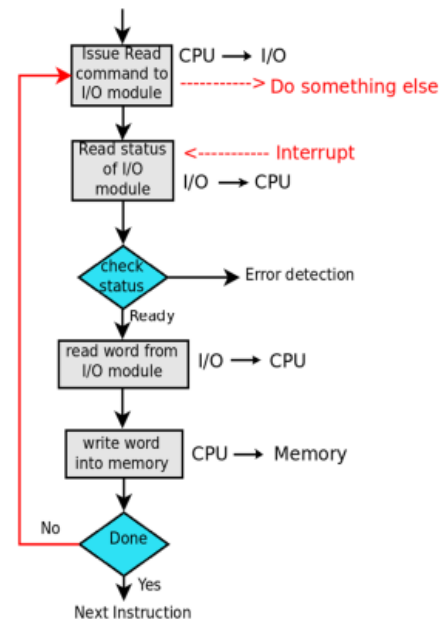
2. प्रोग्राम आय/ओ

प्रोग्राम केलेल्या आय/ओ मध्ये, आय/ओ ऑपरेशनवर संपूर्ण कंट्रोल ठेवण्यासाठी सीपीयू एक प्रोग्राम चालवतो. CPU आणि I/O मॉड्यूल कंट्रोल, चाचणी, वाचन आणि लेखन आदेशांद्वारे संवाद साधतात. सीपीयू आय/ओ मॉड्यूलद्वारे डिव्हाइसची स्थिती जाणतो आणि डेटा ट्रान्समिशन करतो. I/O मॉड्यूलमध्ये दुसरी इंस्ट्रक्शन प्रसारित करण्यापूर्वी हे I/O ऑपरेशन पूर्ण होईपर्यंत प्रतीक्षा करते. कंट्रोल कमांड चालू होते आणि एक्सटर्नल उपकरणाला निर्दिष्ट कार्य निर्देशित करते.

चाचणी कमांड आय/ओ मॉड्यूल आणि त्याच्या एक्सटर्नलची स्थिती तपासते. एक्सटर्नल प्रदेशातील डेटा आय/ओ मॉड्यूलच्या अंतर्गत बफरमध्ये संग्रहित केला जाऊ शकतो. जेव्हा सीपीयूला डेटा बसवर माहिती ठेवण्याची आवश्यकता असते, तेव्हा तो आय/ओ मॉड्यूलला वाचन आदेश जारी करतो. लेखन आदेश I/O मॉड्यूलला परिघावर माहिती पाठवण्याची इंस्ट्रक्शन देतो.



आकृती. 5.11: फ्लोचार्ट प्रोग्राम केलेले I/O साठी



आकृती. 5.12: इंटरप्ट ड्रिव्हन I/O फ्लो

आकृतीमध्ये. 5.11, एक फ्लोचार्ट प्रोग्राम केलेले I/O कार्यपद्धती दर्शवितो. आय/ओ मॉड्यूल सी. पी. यू. कडून वाचन आदेश प्राप्त करते आणि नंतर प्रोसेसरला एक्सटर्नल स्थिती पाठवते. जर I/O डेटा ट्रान्समिशनसाठी तयार असेल, तर I/O मॉड्यूल CPU ला 8 किंवा 16 बिट डेटा पाठवते. सीपीयू मेमोरीमध्ये डेटा पाठवतो. जोडलेले एक्सटर्नल उपकरण डेटा पाठवत नाही तोपर्यंत CPU वेळोवेळी I/O मॉड्यूल तपासते.

3. इंटरप्ट ड्रिव्हन I/O

CPU I/O मॉड्यूलला READ कमांड देते आणि CPU इंटरप्ट ड्रिव्हन I/O दरम्यान इतर काही काम करते. जेव्हा जेव्हा I/O मॉड्यूल जोडलेले एक्सटर्नल



स्कॅन करा

कीबोर्ड आणि डिस्प्लेसह .

8279 नियंत्रकाच्या

इंटरफेससाठी

डिव्हाइस ऑपरेशनसाठी तयार असते तेव्हा एक्सटर्नल डिव्हाइस I/O मॉड्यूलला इंटरप्ट सिग्नल पाठवते, I/O ते प्रोसेसरला पाठवते.

5.4.3 कीबोर्ड आणि डिस्प्ले डिव्हाइसेस

8279 इंटीग्रेटेड सर्किट हे इंटेलने 8085/8086/8088 मायक्रोप्रोसेसरसह कीबोर्ड आणि डिस्प्ले डिव्हाइसेस एकत्रित करण्यासाठी डिझाइन केलेले कीबोर्ड/डिस्प्ले नियंत्रक आहे.

1. कीबोर्ड

कीबोर्डमध्ये 64 किज, 8 रिटर्न लाईन्स, RL0 ते RL7 क्रमांकित आहेत, तसेच अतिरिक्त इनपुट म्हणून शिफ्ट आणि कंट्रोल/स्ट्रोब आहेत. कीबोर्ड माट्रिक्सचे स्तंभ परतीच्या लाइनद्वारे तयार केले जातात. किल्ली स्वयंचलितपणे रद्द केल्या जातात. कीबोर्डमध्ये 2 की लॉकआउट आणि N की रोलओव्हर मोड आहेत. याव्यतिरिक्त, कीबोर्डमध्ये 8x8 फर्स्ट-इन फर्स्ट-आउट (FIFO) रॅम आहे. स्कॅन कीबोर्ड मोडमध्ये, FIFO 8 प्रमुख कोड तसेच शिफ्ट आणि कंट्रोल कीजची स्थिती सेटयित करू शकते. जेव्हा FIFO ची रजिस्टर आढळते, तेव्हा 8279 एक व्यत्यय सिग्नल तयार करते. जेव्हा कळफलक सेन्सर माट्रिक्स मोडमध्ये असतो, तेव्हा 64 स्विचेसची स्थिती FIFO RAM मध्ये जतन केली जाते. कोणत्याही स्विचची स्थिती बदलल्यास प्रोसेसरला व्यत्यय आणण्यासाठी 8279 IRQ (व्यत्यय विनंती) वाढवते.



स्कॅन करा
8279 कंट्रोलरच्या
ब्लॉक आकृतीसाठी

3. डिस्प्ले

प्रदर्शनावरील आउटपुट लाइन A0 ते A3 आणि B0 ते B3 मध्ये विभागल्या जातात. आउटपुट लाइन स्कॅन लाइनसह एकतर 8 लाइन म्हणून किंवा बहुविध प्रदर्शनासाठी 4 लाइनचे दोन गट म्हणून वापरल्या जातात. उदाहरणार्थ, 7-सेगमेंट डिस्प्लेमधील प्रत्येक LED कॅथोड त्याच टर्मिनलशी जोडलेला असतो. डिस्प्ले भागामध्ये 16x8 डिस्प्ले रॅम आहे. CPU मध्ये डिस्प्ले रॅममधील कोणत्याही ठिकाणी वाचण्याची आणि लिहिण्याची क्षमता असते.

युनिट सारांश

- इंटरफेस हा दोन घटकांमधील संवादाचा मार्ग आहे. इंटरफेसिंग हे दोन प्रकारचे असते, मेमोरी इंटरफेसिंग आणि आय. ओ. (इनपुट-आउटपुट) इंटरफेसिंग.
- मेमोरी इंटरफेसिंगमध्ये, इंस्ट्रक्शन अंमलबजावणी दरम्यान, प्रोसेसर वाचन आणि लेखन कार्यासाठी मेमोरीशी संवाद साधतात. आय. ओ. इंटरफेसिंगमध्ये, प्रोसेसर आय. ओ. मॉड्यूलसद्वारे आय. ओ. उपकरणांशी संवाद साधतात.
- मेमोरीचे वर्गीकरण रजिस्टर, कॅशे, यादृच्छिक प्रवेश मेमोरी (प्राथमिक मेमोरी) केवळ वाचन मेमोरी आणि दुय्यम मेमोरीमध्ये केले जाते.
- कॅशेचे दोन प्रकारात वर्गीकरण केले जाते: 'राईट-थ्रू' कॅशे आणि 'राईटबॅक' कॅशे. "राईट-थ्रू" कॅशे एकाच वेळी कॅशे आणि मुख्य मेमोरी ऍड्रेसमध्ये बदल करते.
- राईटबॅक कॅशेमध्ये, फक्त कॅशे स्थान अद्ययावत केले जाते आणि संबंधित फ्लॅग बिट अद्ययावत म्हणून चिन्हांकित केले जाते; हा बिट सामान्यतः गलिच्छ किंवा सुधारित बिट म्हणून ओळखला जातो. जेव्हा हा ब्लॉक काढून टाकला जातो, तेव्हा तो मेमोरी मध्ये साठवला जातो. ही पद्धत वापरणाऱ्या कॅशेना 'राईटबॅक' किंवा 'कॉपी बॅक' कॅशे म्हणतात.
- रॅम (रँडम ऍक्सेस मेमोरी) डेटा आणि प्रोग्राम्स वापरण्यापूर्वी तात्पुरते सेटयित करते. सेट आणि पुनर्संचयित तर्कशास्त्राचा वापर मेमोरी पेशींमध्ये डेटा सेटयित करण्यासाठी केला जातो.
- कॅशे मेमोरी डिझाइन करण्यासाठी SRAM चा वापर केला जातो. प्रत्येक SRAM सेल परिपथाला सहा ट्रान्झिस्टरची आवश्यकता असते. SRAM मध्ये, डेटा वेळोवेळी अद्ययावत करण्याची आवश्यकता नसते.
- DRAM ला प्राथमिक किंवा मुख्य मेमोरी म्हणूनही ओळखले जाते. प्रत्येक DRAM मेमोरी सेल केवळ एक ट्रान्झिस्टर आणि एक कॅपेसिटरने बनलेला असतो. या मेमोरी सेल नियमित अंतराने आपोआप ताजेतवाने होतात.
- रॉम (रीड ओन्ली मेमोरी) ही एक अस्थिर नसलेली मेमोरी आणि डेटा कायमस्वरूपी संग्रहित आहे. या मेमोरीमध्ये प्रणाली प्रोग्राम असतात.
- सेकंडरी मेमोरी मध्ये, डेटा संलग्न मेमोरी ब्लॉकमध्ये वाटप केला जाऊ शकतो आणि संलग्न मेमोरी ब्लॉकमधून डेटा हटविला जाऊ शकतो. ही अस्थिर नसलेली मेमोरी आहे, डेटा कायमस्वरूपी संग्रहित केला जाऊ शकतो.
- संदर्भस्थळाचे तत्त्व असे सांगते की प्रोग्रॅमच्या विशिष्ट फील्डतील अनेक इंस्ट्रक्शन कालांतराने वारंवार अंमलात आणल्या जातात.
 - मुख्य मेमोरीचे ब्लॉक थेट, फुल्ली असोसिएटिव्ह मॅपिंग आणि सेट-असोसिएटिव्ह मॅपिंग द्वारे कॅशे मेमोरीमध्ये माप केले जाऊ शकतात.
 - एकंदरीत, असोसिएटिव्ह मॅपिंग इतर सर्वापेक्षा चांगली कामगिरी करते, परंतु त्याची अंमलबजावणी महागडी आहे. म्हणून, सामान्य पद्धतींमध्ये सेट-असोसिएटिव्ह मॅपिंग ला प्राधान्य दिले जाते.

- एक्सटर्नल उपकरणांसह प्रोसेसरचे इंटरफेस तीन प्रकारे केले जाऊ शकते: मेमोरी/(आय/ओ) माप केलेले आय/ओ, प्रोग्राम केलेले आय/ओ, इंटरप्ट-ड्रिव्हन आय/ओ. 8279 इंटिग्रेटेड सर्किट एक कीबोर्ड/डिस्प्ले कंट्रोलर आहे ज्याची रचना इंटेलने 8085/8086/8088 मायक्रोप्रोसेसरसह कीबोर्ड आणि डिस्प्ले डिव्हाइसेस एकत्रित करण्यासाठी केली आहे.

स्वाध्याय

बहुपर्यायी प्रश्न

प्रश्न 5.1 RAM म्हणजे काय?

(अ) रीड ऍक्सेस मेमोरी (ब) रँडम एडेड मेमोरी (क) रीड अँनालॉग मेमोरी (ड) रँडम ऍक्सेस मेमोरी

प्रश्न 5.2 खालीलपैकी कोणते मेमोरी डिव्हाइस, मुख्यतः वेगाच्या बाबतीत, कॅशे मेमोरीशी बरेच साम्य आहे?

(अ) SRAM (ब) DRAM (क) EEPROM (ड) फ्लॅश मेमोरी

प्रश्न 5.3 खालीलपैकी कोणती संगणकातील स्थिर, अस्थिर नसलेली आणि कायमस्वरूपी मेमोरी आहे?

(अ) CDROM (ब) CPU (क) ROM (ड) RAM

प्रश्न 5.4 डिजिटल कॅमेरा _____ मेमोरी वापरतो.

(अ) फ्लॅश (ब) मुख्य (क) कॅशे (ड) आभासी

प्रश्न 5.5: सर्वात लहान आणि सर्वात जास्त स्टोरेज युनिट ओळखा?

(अ) जीबी आणि टीबी (ब) जीबी आणि एमबी (क) एमबी आणि टीबी (ड) केबी आणि टीबी

प्रश्न 5.6 _____ हा दुय्यम मेमोरीचा प्रकार नाही?

(अ) सॉलिड स्टेट ड्राइव्ह (ब) हार्ड डिस्क (क) रॅम (ड) युएसबी पेन ड्राइव्ह

प्रश्न 5.7 खोटी विधाने ओळखणे:

(1) प्रोग्रामेबल रीड ओन्ली मेमोरी एकदा लिहिली जाते आणि विशेष PROM प्रोग्रामर वापरून प्रोग्राम केली जाते.

(2) अतिनील प्रकाशाद्वारे EPROM मेमोरी प्रोग्राम केली जाऊ शकते आणि नष्ट केली जाऊ शकते.

(3) विद्युत वोल्टेज वापरून EEPROM डेटा मिटवतो.

(4) फ्लॅश मेमोरीमध्ये, डेटा पुन्हा लिहिला जाऊ शकतो.

(अ) 1,2,3 आणि 4 (ब) 1,2 आणि 3 (क) 1 आणि 2 (ड) यापैकी काहीही नाही

प्रश्न 5.8 वैयक्तिक संगणकाच्या मुख्य मेमोरी मध्ये _____ असते.

(अ) RAM आणि ROM दोन्ही (ब) कॅशे मेमोरी (क) केवळ रॅम (ड) केवळ रॅम

प्रश्न 5.9 _____ रॅमचे प्रकार उपलब्ध आहेत?

(अ) चार (ब) तीन (क) दोन (ड) पाच

प्रश्न 5.10 प्रणालीच्या बूट सेक्टर फाइल्स _____ मध्ये संग्रहित केल्या जातात.

(अ) कॅशे (ब) रीड ओन्ली मेमोरी (क) रँडम ऍक्सेस मेमोरी (ड) रजिस्टर

प्रश्न 5.11 डिव्हाइसेस आणि मेमोरी _____ I/O साठी स्वतंत्र अँड्रेस स्पेस वापरून इंटरफेस केली जातात.

(अ) प्रोग्राम (ब) इंटरप्ट ट्रिग्नर (क) मेमोरी मापड (ड) आय/ओ मापड

प्रश्न 5.12 _____ इंटिग्रेटेड सर्किट हे इंटेलने डिझाइन केलेले कीबोर्ड/डिस्प्ले नियंत्रक आहे.

(अ) 8085 (ब) 8086 (क) 8088 (ड) 8279

प्रश्न 5.13 8255 IC मध्ये

(अ) 12 (ब) 24 (क) 48 (ड) 64

प्रश्न 5.14 पूर्वनिर्धारित रोटेशनल विलंबता ही रोटेशन वेळेच्या _____ वेळा आहे.

(अ) 1/2 (ब) 1/4 (क) 2 (ड) 4

प्रश्न 5.15 8255 साठी कोणती कार्यपद्धती नाही?

(अ) आय/ओ मोड (ब) बीएसआर मोड (क) एमएसबी मोड (ड) मोड 0

बहुपर्यायी प्रश्नांचे उत्तरे

5.1	(ड)	5.2	(अ)	5.3	(क)	5.4	(अ)	5.5	(ड)	5.6	(क)
5.7	(ड)	5.8	(अ)	5.9	(क)	5.10	(ब)	5.11	(ड)	5.12	(ड)
5.13	(ब)	5.14	(अ)	5.15	(क)						

लघु आणि दीर्घ उत्तर प्रकार

प्रश्न श्रेणी-I

प्रश्न 5.1 डायनॅमिक रॅममध्ये साठवलेली माहिती वेळोवेळी रीफ्रेश करण्याची गरज का आहे?

प्रश्न 5.2 कीबोर्डसह इंटरफेस करणे हे डिस्प्लेसह इंटरफेस करण्यापेक्षा कसे वेगळे आहे?

प्रश्न 5.3: प्रवेश वेळ, मेमोरी खर्च आणि क्षमता वेगवेगळ्या मेमोरी प्रकारांसाठी कशी बदलते?

प्रश्न 5.4 सेट-संबंधित मारपिंग, सहयोगी मारपिंग आणि थेट मारपिंगचे फायदे/तोटे यांची तुलना करा.

प्रश्न 5.5 प्रत्येक कॅशे मेमोरी मारपिंग दृष्टीकोन मुख्य मेमोरी पत्त्यांना फील्ड्स म्हणून हाताळतो. अशी फील्ड निर्दिष्ट करा.

प्रश्न 5.6 SRAM आणि DRAM मेमोरी मध्ये किती ट्रान्झिस्टर वापरले जातात?

प्रश्न 5.7 प्रोग्राम केलेले आणि मेमोरी माप केलेले I/O तुलना करा.

प्रश्न 5.8 RAM आणि ROM मध्ये काय फरक आहे?

प्रश्न 5.9 संगणक मेमोरी प्रणाली पदानुक्रम म्हणून का तयार केल्या जातात?

प्रश्न 5.10 DRAM ची स्टोरेज क्षमता SRAM पेक्षा जास्त का असते?

प्रश्न 5.11 कॅशे क्षमता आणि हिट रेटमध्ये काय संबंध आहे?

प्रश्न 5.12 कॅशेची असोसिएटिव्हिटी वाढवल्याने सामान्यतः त्याचा हिट रेट का वाढतो?

प्रश्न 5.13 संदर्भ स्थान म्हणजे काय?

प्रश्न 5.14 स्थानिक स्थान आणि लौकिक स्थान यामधील दोन भेद स्पष्ट करा.

प्रश्न 5.15 Write Through आणि Writeback Cache मध्ये काय फरक आहे?

प्रश्न श्रेणी-II

प्रश्न 5.16 वेगवेगळ्या अनुप्रयोगांसाठी विविध मेमोरी तंत्रज्ञानाचा वापर कसा केला जातो याचे वर्णन करा. फ्लॅश मेमोरी तंत्रज्ञान, PROM, EPROM आणि EEPROM पासून ROM कसा वेगळा आहे?

प्रश्न 5.17 SRAM आणि DRAM तंत्रज्ञानामधील फरक सांगा.

प्रश्न 5.18 कॅशेच्या रेषेची लांबी का वाढते?

(i) च्या रेषेची लांबी वाढवल्याने त्याचा हिट रेट का वाढतो?

(ii) कॅशेचा हिट रेट वाढला तरी कधीकधी कॅशे असलेल्या प्रणालीची कार्यक्षमता कमी होते का?

(iii) फटका बसण्याचा दर कमी होऊ शकतो का?

प्रश्न 5.19 मेमोरी इंटरफेसिंग आणि आयओ इंटरफेसिंगमध्ये काय फरक आहे? बाह्य उपकरणांमधून सीपीयूमध्ये डेटा ट्रान्सफर करण्याच्या कार्यपद्धती स्पष्ट करा.

प्रश्न 5.20 कीबोर्ड आणि डिस्प्ले डिव्हाइसेसशी कसे संवाद साधायचे याचे तपशीलवार वर्णन करा.

प्रश्न 5.21 कॅशे मेमोरी मध्ये मेमोरी ब्लॉक्सचे मार्पिंग करण्यासाठी थेट, सहयोगी आणि सेट-असोसिएटिव्ह मार्पिंग पद्धतींचे वर्णन करा.

संख्यात्मक प्रश्न

प्रश्न 5.22 सेट-संबंधित मेमोरीचे 128 ब्लॉक चार ब्लॉक संचांमध्ये विभागले जातात. मुख्य मेमोरीतील 16,384 ब्लॉक्सपैकी प्रत्येकामध्ये 256 आठ-बिट वर्ड असतात. (i) मुख्य मेमोरी ऍड्रेससाठी किती बिट्स आवश्यक आहेत?

(ii) 'टॅग', 'सेट' आणि 'वर्ड' फील्ड?

[उत्तर: (i) 22 बिट्स (ii) टीएजी = 9 बिट्स, सेट = 5 बिट्स, वर्ड = 8 बिट्स]

प्रश्न 5.23 जर कॅशे 2-वे, 4-वे किंवा 8-वे सेट-असोसिएटिव्ह असेल आणि त्याची क्षमता 16 KB असेल तर त्याच्या रेषेची लांबी किंवा ब्लॉकचा साईझ 128 बाइट असेल तर त्यात किती सेट असतील?

[उत्तर: (i) 64 (ii) 32 (iii) 16]

प्रश्न 5.24 8-बिट रुंदीच्या रॅम चिपमध्ये 1024 वर्ड साठवले जाऊ शकतात (1Kx8). 1Kx8 रॅमला 16Kx16 रॅममध्ये रूपांतरित करण्यासाठी, सक्षम रेषेसह किती 2x4 डीकोडर्सची आवश्यकता आहे?

[उत्तर: 5]

प्रश्न 5.25 A प्रणालीमध्ये 1 जीबी मुख्य मेमोरी आहे आणि 32-बिट मेमोरी ऍड्रेसिंग वापरते. त्यात ब्लॉक-सेट-असोसिएटिव्ह पद्धतीने आयोजित 8M-बाइट कॅशे आहे,



स्कॅन करा
जेम5 सिम्युलेटरच्या
स्थापनेच्या पायऱ्या

ज्यामध्ये प्रति सेट 4 ब्लॉक आणि प्रति ब्लॉक 64 बाइट आहेत. मेमोरी ऍड्रेसची फील्ड कोणती आहेत?

[उत्तर: टॅग = 11, सेट = 15, वर्ड = 6]

प्रश्न 5.26 64K 32-बिट शब्दांचा समावेश असलेली मुख्य मेमोरी. यात प्रति ब्लॉक 16 शब्दांसह 2के वर्ड डायरेक्ट-माप कॅशे देखील आहे. समजा CPU हा 32-बिट वर्ड वापरण्यासाठी 16-बिट हेक्साडेसिमल ऍड्रेस ABCD निर्माण करतो, हा वर्ड ज्याच्याशी जुळतो तो कॅशे ब्लॉक क्रमांक दशांश मध्ये निर्दिष्ट करा.

[उत्तर: 60]

प्रश्न 5.27 एका डिस्कमध्ये 10 डेटा रेकॉर्डिंग पृष्ठभाग असतात, प्रत्येकी 4096 ट्रॅक असतात. जर ट्रॅक 128 सेक्टरमध्ये विभागले गेले असतील आणि प्रत्येक सेक्टरमध्ये अनुक्रमे 256 बाइट रजिस्टरवले गेले असतील, तर डिस्कची एकूण क्षमता किती असेल?

[उत्तर: डिस्क क्षमता = $10 \times 4096 \times 128 \times 256$]

प्रात्यक्षिक:

उद्देश: जेम 5 सिम्युलेटरमध्ये, x86 आणि अल्फा सारख्या दोन इन्स्ट्रक्शन सेट आर्किटेक्चरचे (IAS) अनुकरण करा. प्रत्येक IAS च्या विजेचा वापर आणि कामगिरीची तुलना करा. विविध कॅशे आणि DRAM मेमोरी मॉडेलचे संरचना मापदंड बदला आणि हिट आणि मिसच्या संख्येचा प्रणालीच्या वीज वापर आणि कार्यक्षमतेवर कसा परिणाम होतो हे ठरवा.



स्कॅन करा
जेम5 शिकवणीसाठी

साधने: Gem 5 सिम्युलेटर सिद्धांत: Gem5 सिम्युलेटर M5 आणि GEMS सिम्युलेटरने बनलेले आहे. Mम5 मध्ये भिन्न ISA, भिन्न CPU मॉडेलस आणि एक सिम्युलेशन प्रणाली आहे जी बदलली जाऊ शकते. GEMS ही एक लवचिक मेमोरी प्रणाली आहे ज्यात अनेक कॅशे सुसंगतता पद्धती आणि परस्पर जोडणी मॉडेल आहेत. Gem5 हे ARM, ALPHA, MIPS, पॉवर, SPARC आणि x86 सह कार्य करते, जे सर्व व्यावसायिक ISA आहेत.



स्कॅन करा
जेम5 पॅरामीटर केलेले
पर्याय शिकण्यासाठी

प्रक्रिया: Gem5 हे मोठ्या प्रमाणावर वापरले जाणारे चक्र-अचूक संगणक आर्किटेक्चर सिम्युलेटर आहे. हा एक मुक्त सोर्स प्रकल्प आहे जो संशोधक आणि विकासकांना सॉफ्टवेअर प्लॅटफॉर्मवर अनुकरण करून विविध संगणक संरचना आणि प्रणाली-स्तरीय डिझाईन्स शोधण्यास सक्षम करतो.

Gem5 च्या स्थापनेच्या पायऱ्या ऑपरेटिंग सिस्टम आणि Gem5 च्या स्थापित करण्याच्या विशिष्ट आवृत्तीवर अवलंबून बदलू शकतात. उबंटू 22.04 वर Gem5 स्थापित करण्यासाठी चरण येथे आहेत:

स्टेप 1 खालील आदेश चालवून जेम 5 तयार करण्यासाठी आवश्यक डिपेंडन्सी स्थापित करा:

```
sonal@sonal-virtual-machine: ~/gem5
sonal@sonal-virtual-machine:~$ sudo apt install build-essential git m4 scons zlibg
zlibg-dev libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev pyt
hon3-dev python-is-python3 libboost-all-dev pkg-config
```

स्टेप 2 खालील आदेश चालवून GitHub पासून gem5 रिपॉझिटरी क्लोन करा:

```
sonal@sonal-virtual-machine: ~/gem5
sonal@sonal-virtual-machine:~$ git clone https://gem5.googlesource.com/public/gem5
```

स्टेप 3 खालील आदेश चालवून gem5 निर्देशिका प्रविष्ट करा.

```
sonal@sonal-virtual-machine: ~/gem5
sonal@sonal-virtual-machine:~ $ cd gem5/
```

स्टेप 4 खालील आदेश वापरून gem5 तयार करा

```
sonal@sonal-virtual-machine: ~/gem5
sonal@sonal-virtual-machine:~/gem5$ scons build/NULL/gem5.debug PROTOCOL=Garnet_standalone
--linker=gold
```

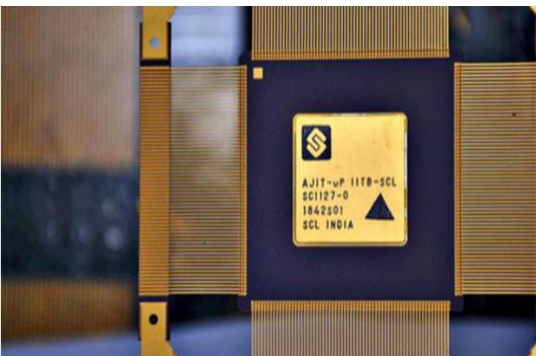
स्टेप 5 आज्ञा चालवून तुम्ही स्थापनेची चाचणी करू शकता:

```
sonal@sonal-virtual-machine: ~/gem5
sonal@sonal-virtual-machine:~ $ ./build/NULL/gem5.debug configs/example/garnet_synth_traffic.py --
num-cpus=16 --num-dirs=16 --network=garnet --topology=Mesh_XY --mesh-rows=4 --caches --l2cache --ruby
--sim-cycles=10000 --synthetic=uniform_random --injectionrate=0.1
```

खालील आदेश ओळ पर्यायांचा वापर विविध कॅशे मेमोरी संरचनांचा शोध घेण्यासाठी केला जाऊ शकतो:

```
--caches
--l2cache
--num-dirs NUM_DIRS
--num-l2caches NUM_L2CACHES
--num-l3caches NUM_L3CACHES
--l1d_size L1D_SIZE
--l1i_size L1I_SIZE
--l2_size L2_SIZE
--l3_size L3_SIZE
--l1d_assoc L1D_ASSOC
--l1i_assoc L1I_ASSOC
--l2_assoc L2_ASSOC
--l3_assoc L3_ASSOC
--cacheline_size CACHELINE_SIZE
--ruby
-m TICKS, --abs-max-tick TICKS
```

अधिक जाणून घ्या भारतीयांनी केलेले नवकल्पना



माधव देसाई आणि त्यांच्या चमूने इलेक्ट्रॉनिक्स आणि माहिती तंत्रज्ञान मंत्रालयाच्या अर्थसहाय्याने आयआयटी बॉम्बे येथे AJIT प्रोसेसर विकसित केले आहे (MeitY).

AJIT हा एक मध्यम आकाराचा प्रोसेसर आहे, जो इंटेलच्या झिऑन आणि इतर लॅपटॉप प्रोसेसरपेक्षा वेगळा आहे. याचा वापर सेट-टॉप बॉक्समध्ये, ट्रॅफिक लाइट ड्रायव्हर म्हणून, ऑटोमेशन सिस्टमसाठी

कंट्रोल पॅनेल म्हणून किंवा अगदी रोबोटिक सिस्टममध्येही केला जाऊ शकतो. जेव्हा AJIT मोठ्या प्रमाणात तयार केली जाईल, तेव्हा किंमत खूप कमी असेल. तो प्रति घड्याळ चक्र एक आदेश चालवू शकता आणि 70 आणि 120 मेगाहर्ट्झ दरम्यान घड्याळ गती, जे बाजारात त्याच्या प्रतिस्पर्ध्यांना सुमारे समान आहे चालवू शकता [7].

आधुनिक जीवनशैलीमध्ये ध्यानाचे महत्त्व ध्यानधारणा

हायपोथॅलेमस-पिट्यूटरी-एड्रेनल (एचपीए) एक्सिसवर लक्षणीय परिणाम करते, मेंदू-शरीराचे सर्किट जे तणावास शरीराच्या प्रतिसादात महत्त्वाची भूमिका बजावते [8]. ध्यानामुळे ऊर्जेची पातळी वाढते आणि रोगप्रतिकारक शक्ती वाढते, ज्यामुळे शरीर रोगांशी लढू शकते. यामुळे विश्रांतीचा प्रतिसाद आणि इतर मानसिक शारीरिक प्रक्रिया घडतात.



मनाचे शरीरावर विलक्षण कन्ट्रोल असते. एखाद्या व्यक्तीची मानसिक स्थिती नाडीचा वेग, रक्तदाब आणि जैवरासायनिक रेणूंचे उत्पादन, मानवी संप्रेरक आणि न्यूरोट्रांसमीटर यासारख्या शारीरिक कार्यांवर परिणाम करू शकते. ध्यानधारणा, इतर निरोगी-जीवन पद्धतींच्या संयोगाने, तणावाशी संबंधित रोग आणि नैराश्याच्या मनःस्थिती विकारांसाठी औषध-मुक्त उपचार होण्याची क्षमता आहे [9,10].

ध्यान हा आध्यात्मिक आणि मानसिक अभ्यासाचा एक प्रकार आहे. हे प्रेम, आनंद आणि शांतीशी जोडले जाण्यास मदत करते. ध्यान करताना, एखाद्याला सखोल विश्रांतीचा अनुभव येतो. ध्यानधारणेचे अनेक फायदे आहेत, ज्यात शांत मन, वाढीव एकाग्रता आणि आत्म-कन्ट्रोल, उच्च एकाग्रता शक्ती, वाढीव उत्पादकता, आत्मविश्वास, आत्म-जागरूकता आणि करुणा यांचा समावेश आहे. हे मन आणि शरीराला देखील बरे करते आणि उर्जेच्या आतील स्रोताशी जोडते. ध्यानधारणा प्राचीन भारतात शोधली जाऊ शकते. दरवर्षी, वाढत्या संख्येने शास्त्रज्ञ ध्यानाच्या आरोग्यविषयक फायद्यांवर संशोधन प्रकाशित करतात. जसजसे अधिक संशोधन प्रकाशित होत जाते, तसतसे अधिक व्यक्तींना ध्यान करण्यास प्रवृत्त केले जाते. ध्यानाचे सर्वात जुने संदर्भ सुमारे 1500 B.C. पासून भारतीय ग्रंथांमध्ये आढळतात. तथापि, भारताच्या वैदिक ग्रंथांमध्ये असे म्हटले आहे की मानवतेच्या सुरुवातीपासूनच ध्यान अस्तित्वात आहे. भारताव्यतिरिक्त, अनेक प्राचीन समाजांनी ध्यानाकडे आध्यात्मिक वाढीसाठी एक शक्तिशाली साधन म्हणून पाहिले.

ध्यानधारणा तुम्हाला शारीरिक, मानसिक आणि भावनिक आरोग्य राखण्यास मदत करू शकते. ध्यानधारणा ही तुमच्या नियमित दिनचर्येत समाविष्ट करण्यासाठी सोपी आहे. सातत्यपूर्ण ध्यानधारणेद्वारे, तुम्ही एक सखोल आंतरिक बदल अनुभवाल जो तुमच्या सभोवतालच्या इतरांना दिसून येईल, कारण तुम्ही उत्सर्जित करता ती सकारात्मक ऊर्जा त्यांना जाणवेल.

ध्यान तंत्रे आज विविध प्रकारात उपलब्ध आहेत. तुमच्या व्यक्तिमत्त्वाशी आणि आवडीनिवडीशी जुळणारा पर्याय निवडा. कडक वेळापत्रकाला बांधील न राहता तुम्ही लगेच नियमित ध्यानधारणा करू शकता. ध्यानधारणेसाठी थोडा वेळ आणि शांत जागा बाजूला ठेवा. एकदा तुम्ही ते पूर्ण केले की, तुम्हाला ते पुढच्या वेळी करायचे आहे. ध्यान करताना, टाइमर सेट करणे आणि शक्य तितके संभाव्य व्यत्यय काढून टाकणे उपयुक्त ठरते. प्रत्येकामध्ये ध्यान करण्याची जन्मजात क्षमता असते, जशी प्रत्येकामध्ये चालण्याची जन्मजात क्षमता असते. ही प्रॅक्टीसची बाब आहे. ध्यानधारणा आणि लक्षपूर्वक व्यायाम केल्याने तुम्हाला तुमच्या जीवनात शांततापूर्ण आणि ठाम ऊर्जा जाणवण्यास मदत होऊ शकते.

संदर्भ आणि सुचविलेले वाचन

- [1] प्रा. इंद्रनील सेनगुप्ता आणि प्रा. कमलिका दत्ता यांचा एनपीटीईएल अभ्यासक्रम, संगणक वास्तुकला आणि संघटना, आयआयटी खरगपूर, 2017.
<https://archive.nptel.ac.in/cours/106/105/106105163/> (last accessed: Jan 07, 2023)
- [2] निकोलस कार्टर, कॉम्प्युटर आर्किटेक्चर, शॉम्स आउटलाइन, 2002.
 प्रा. मधु मुत्यम यांचा एनपीटीईएल अभ्यासक्रम, संगणक वास्तुकला, आयआयटी मद्रास, 2015.
<https://archive.nptel.ac.in/cours/106/106/106106134/> (last accessed: Jan 07, 2023)
- [4] कार्ल हमाकर, इवोन्को व्रॅनेसिक, सफवत झाकी आणि नरैग मंजिकियन, संगणक संस्था आणि एम्बेडेड सिस्टीम्स. माकग्रा-हिल उच्च शिक्षण, 2011.
- [5] एम. मॉरिस मनो, संगणक प्रणाली वास्तुकला. प्रेंटिस-हॉल, इंक., तिसरी आवृत्ती.
<https://poojavaishnav.files.wordpress.com/2015/05/mano-m-m-computer-systemarchitecture.pdf> (last accessed: Jan 07, 2023)
- [6] विल्यम स्टॅलिंग्स, कॉम्प्युटर ऑर्गनायझेशन अँड आर्किटेक्चर डिझायनिंग फॉर परफॉर्मन्स, 10 वी आवृत्ती, 2016.
- [7] 'मेड इन इंडिया' मायक्रोप्रोसेसर ए. जे. आय. टी. चे स्वागत आहे,
<https://www.iitb.ac.in/en/researchhighlight/welcome-ajit-%E2%80%98made-india%E2%80%99-microprocessor> (last accessed: Jan 07, 2023)
- [8] न्यूरोकेमिकल्सवर ध्यानाचा प्रभाव, <https://sahajaonline.com/science-health/mentalhealth-well-being/neurochemicals/evidence-of-meditations-impact-on-neurotransmittersneurohormones/> (last accessed: Jan 07, 2023)
- [9] विल्यम सी. डाउब आणि चार्ल्स ई. जाकोबशे, बायोकेमिकल इफेक्ट्स ऑफ मेडिटेशन: अ लिटरेचर रिव्ह्यू. स्कॉलरली अंडरग्रॅज्युएट रिसर्च जर्नल एट क्लार्क: ब्लॉक. 1, अनुच्छेद 10, 2015.
<https://commons.clarku.edu/surj/vol1/iss1/10> (last accessed: Dec 30, 2022)
- [10] अँडम कॉन्क्झ, इसोल्ट डेमेट्रोव्हिक्स आणि इसोफिया के. टाकास, ध्यानधारणा हस्तक्षेप जोखीम असलेल्या नमुन्यांची कोर्टिसोल पातळी कार्यक्षमतेने कमी करतात: एक मेटा-विश्लेषण. आरोग्य मानसशास्त्र पुनरावलोकन, 15:1, 56-84, 2021. माहिती: 10.1080/17437199.2020.1760727.

पुढील शिक्षणासाठी संदर्भ

प्रत्येक युनिटमध्ये दिलेली उदाहरणे आणि समस्या सोडवल्यानंतर संगणक प्रणाली संघटनेच्या संकल्पना अधिक स्पष्ट होतील. असेंब्ली प्रोग्रामिंग शिकण्यासाठी युनिट-IV ची उदाहरणे आणि संदर्भ दुव्यांचा सराव करा. प्रत्येक अध्यायाच्या शेवटी सुचवलेल्या वाचनासाठी आणि एन. पी. टी. ई. एल. अभ्यासक्रमांसाठी दुवे आहेत. प्रगत अभ्यास आणि संख्यात्मक समस्या सोडवण्यासाठी अतिरिक्त संसाधने खालीलप्रमाणे आहेत:

[1] कार्ल हॅमाकर, संगणक संस्था आणि एम्बेडेड सिस्टीम्स. माकग्रा हिल पब्लिकेशन, 2002. <http://103.62.146.201:8081/jspui/bitstream/1/9025/1/bok>. (अखेरचा प्रवेश: 30 मे 2023)

[2] निकोलस कार्टर, संगणक वास्तुकला. शॉम्स आउटलाइन, 2002.

प्रो. मेमोरी रंजन सारंगी यांचा एनपीटीईएल अभ्यासक्रम, प्रगत संगणक वास्तुकला, आयआयटी दिल्ली, 2021. <https://archive.nptel.ac.in/cours/106/102/106102229> / (शेवटचा प्रवेश: 07 जानेवारी 2023)

[4] प्रा. व्ही. कामकोटी, संगणक संघटना आणि वास्तुकला, आयआयटी मद्रास, 2017 द्वारे एनपीटीईएल अभ्यासक्रम. <https://archive.nptel.ac.in/cours/106/106/106106166> / (last accessed: Jan 07, 2023)

[5] मुहम्मद यझार वाय, एनएसएमचा परिचय.

<https://usermanual.wiki/Document/NASM20Manual.1164426225/view> (शेवटचा प्रवेश: 15 मे 2023)

पूरक माहिती म्हणून, लेखक पुस्तकाच्या प्रकरणांच्या काही विषयांवर, संबंधित संख्यात्मक समस्या आणि व्यावहारिक विषयांवर व्हिडिओ व्याख्याने विकसित करत आहे. व्हिडिओची लिंक लेखकाच्या संकेतस्थळावर <https://nitrr.ac.in/viewdetails.php?q=cse.syadav> उपलब्ध आहे.

सीओ आणि पीओ अटेंन्मेंट टेबल

या अभ्यासक्रमाचे अभ्यासक्रम परिणाम (सीओ) अभ्यासक्रम पूर्ण झाल्यानंतर प्रोग्रॅमच्या परिणामांसह (पीओ) माप केले जाऊ शकतात आणि अंतरांचे विश्लेषण करण्यासाठी पीओच्या प्राप्तीसाठी परस्परसंबंध तयार केला जाऊ शकतो. पी. ओ. च्या प्राप्तीतील अंतरांचे योग्य विश्लेषण केल्यानंतर अंतर दूर करण्यासाठी आवश्यक उपाययोजना केल्या जाऊ शकतात.

सीओ आणि पीओ प्राप्तीसाठी सारणी

युनिट निष्पत्ती	प्रोग्रॅमच्या परिणामांची निष्पत्ती 1-कमकुवत मापिंग, 2- मध्यम मापिंग, 3- मजबूत मापिंग						
	PO1	PO2	PO3	PO4	PO5	PO6	PO7
CO-1							
CO-2							
CO-3							
CO-4							
CO-5							

वरील तक्त्यात भरलेली माहिती अंतर विश्लेषणासाठी वापरली जाऊ शकते.

इंडेक्स

अ

अॅड्रेसिंग मोड 106,107,108,109,111,162

एड्रेस सिक्वेन्सिंग 51

अरीथमेटिक आणि लॉजिक युनिट (ALU) 8,12

एरिथमेटिक इन्स्ट्रक्शन 119, 146,148

अरीथमेटिक आणि लॉजिक शिफ्ट युनिट 30

अरीथमेटिक पाइपलाइन 75

अरे प्रोसेसर 86,87

असेम्बलर 132,133,145

असेम्बलरचे डारेक्टिव 141

असेम्बली लॅंग्वेज प्रोग्रॅम 134

आ

आर्किटेक्चर ऑफ 8086 मायक्रोप्रोसेसर 114

ब

ब्रांच इन्स्ट्रक्शन 82,83,84,151

बस आणि मेमोरी ट्रान्सफर 27

बस संरचना 13

क

कॅशे मेमोरी 7,8,171

कॅशे मेमोरी मारपिंग टेक्निक 172

सेंट्रल प्रोसेसिंग युनिट 101,169

कॉम्प्युटर अरीथमेटिक 58

बेरीज 58

वजाबाकी 58

गुणाकार 58, 61

भागाकार 66

कंट्रोल मेमोरी 52,54,57

कंट्रोल युनिट 52,54,59,62,119

ड

डेटा प्रतिनिधित्व 16

फ्लोटिंग पॉइंट 17

डिजिटल संगणक 2, 12, 25

इ

इवलुएशन ऑफ अरीथमेटिक एक्सप्रेसन 153

इन्स्ट्रक्शन फॉरमाट 105,106

इन्स्ट्रक्शन पाईपलाईन 76, 85

इन्स्ट्रक्शन सेट आर्किटेक्चर 104,105,

सीआयएससी वैशिष्ट्ये 104

आरआयएससी वैशिष्ट्ये 105

इंटरकनेक्शन 10,11

इंटरफेसिंग कीबोर्ड आणि डिस्प्ले डिव्हाइसेस 186

इंटरफेसिंग प्रोसेसर 185

ए

एरर डिटेक्शन कोड 23,24

ओ

ओव्हरफ्लो डिटेक्शन 18,19

ऑपरेशनल मोड 8255 183

आउटपुट युनिट 12

प

प्रोसीजर आणि माक्रो 143

प्रोग्रामेबल एक्सटर्नल इंटरफेस 183

प्रोग्राम आय/ओ 185

फ

फिक्स्ड पॉइंट 69

फिक्स्ड पॉइंट रेप्रेसेंटेशन 69

फ्लोटिंग पॉइंट अरीथमेटिक 70

म

मेमोरी आणि आय/ओ इंटरफेसिंग 169

मेमोरी मानजमेण्ट युनिट (MMU) 179

मेमोरी-माप आय/ओ 185

मेमोरी प्रकार आणि वैशिष्ट्ये 170

मायक्रो ऑपरेशन 3

अरीथमेटिक 31

लॉजिकल 32

शिफ्ट 35

मायक्रोप्रोसेसर 56,104

8085 102,103,104

8086 99,100,103,104

मायक्रोप्रोग्राम कंट्रोल 52,56,57,106

रैंडम एक्सेस मेमोरी (RAM) 171

रीड ओन्ली मेमोरी (ROM) 180

रजिस्टर ट्रान्सफर 25,26,27

आरआयएससी पाइपलाइन 85,105

ल

लोकॅलिटी ऑफ रेफेरेंस 172

लॉजिकल इन्स्ट्रक्शन 150

व

वेक्टर प्रोसेसिंग 86

वॉन-न्युमेन आर्किटेक्चर 12

स

सेकंडरी मेमोरी 181

सॉर्टिंग 156

स्ट्रिंग मानिपुलेशन 154

ह

हार्डवायर्ड कंट्रोल युनिट 55

ह्रस्वार्ड 77,79,80,81,82



संगणक प्रणाली संघटन

डॉ. सोनल यादव

संगणक प्रणाली संघटनेच्या प्रमुख संकल्पना या पुस्तकात समाविष्ट केल्या आहेत. संगणक संरचना, कंट्रोल युनिट डिझाइन, अंकगणितीय कार्ये, मायक्रोप्रोसेसर आर्किटेक्चर, असेंब्ली लॅंग्वेज प्रोग्रामिंग आणि इनपुट-आउटपुट उपकरणांसह मेमरी इंटरफेसिंग या सर्व मूलभूत संकल्पना या पुस्तकात समाविष्ट आहेत. हे पुस्तक सैद्धांतिक ज्ञानाचा व्यावहारिक उपयोगासह मेळ घालते. हे व्हेरिलॉग एचडीएल भाषा, 8086 मायक्रोप्रोसेसर, एनएसएम असेंबलर आणि जेम 5 सिम्युलेटर वापरून अत्याधुनिक प्रयोग देखील सादर करते. अध्यायांमध्ये दिलेले क्यू. आर. कोड स्कॅन करून प्रत्येक विषयाचा पुढील स्तरापर्यंत अभ्यास केला जाऊ शकतो.

हे पुस्तक सी. एस. ई., आय. टी., ई. सी. ई. आणि एम. सी. ए. च्या डिप्लोमा आणि पदवीपूर्व विद्यार्थ्यांसाठी लिहिलेले आहे, जेणेकरून संगणकीय संस्थांच्या सोप्या ते गुंतागुंतीच्या समस्यांचे विश्लेषण आणि निराकरण करण्याची क्षमता बळकट होईल. "अधिक जाणून घ्या" विभागात, उल्लेखनीय भारतीय शोधक तसेच समृद्ध भारतीय वेद ज्ञान आणि मूलभूत तत्त्वे वाचकांना आधुनिक जीवनशैलीमध्ये आपली मौल्यवान तत्त्वे पाळण्यास प्रेरित करण्यासाठी सादर केली आहेत.

मूक वैशिष्ट्ये:

- अभ्यासक्रमाचे परिणाम, प्रोग्रॅम चे परिणाम आणि युनिट परिणामांच्या मॅपिंगशी पुस्तकाची सामग्री संरेखित केलेली आहे.
- प्रत्येक युनिटच्या सुरुवातीला, ते युनिट पूर्ण केल्यानंतर त्याच्याकडून/तिच्याकडून काय अपेक्षित आहे हे विद्यार्थ्यांना समजून घेण्यासाठी शिकण्याचे परिणाम सूचीबद्ध केले आहे.
- पुस्तकामध्ये बरीच अलीकडील माहिती, मनोरंजक तथ्ये, ई-संसाधनांसाठी क्यू. आर. कोड, आय. सी. टी. च्या वापरासाठी क्यू. आर. कोड, प्रकल्प, गट चर्चा इ. प्रदान केली आहेत.
- संतुलित आणि कालक्रमानुसार पुस्तकात समाविष्ट केलेले विषयाचे साहित्य विद्यार्थी आणि शिक्षक साठी केंद्रित आहे.
- विषयांची स्पष्टता सुधारण्यासाठी आकृती, तक्ता आणि सॉफ्टवेअर स्क्रीन शॉट्स समाविष्ट केले आहे.

