Topic 3: Building a CPU

ICT170: Foundations of Computer Systems

Overview

Topics:

- CPU Organisation
- Instruction Execution
- Design Principles of Modern Computers
- Processor Parallelism
- The Microarchitecture Level
- The Instruction Set Architecture

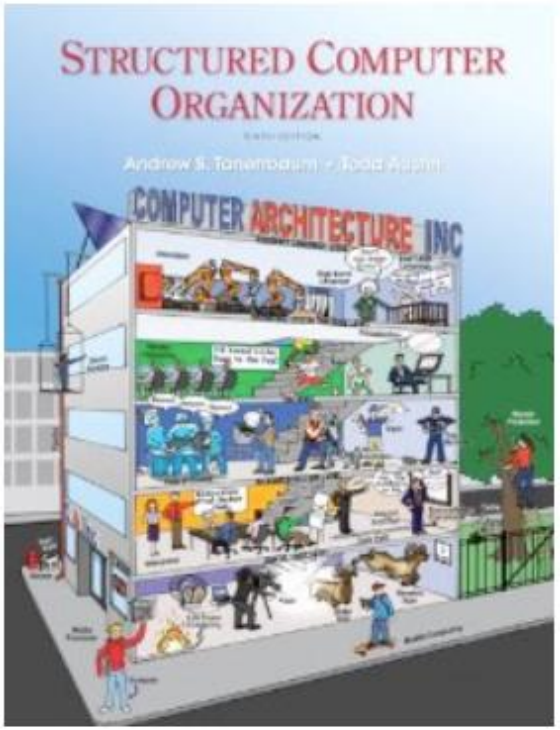
Objectives

In order to achieve the unit learning objectives, on successful completion of this topic, you should be able to:

- Describe the basic architecture of a CPU and how it relates to a simple computer.
- Explain the process of instruction execution in a simple CPU.
- Understand and be able to describe the different styles of CPU-level Parallelism

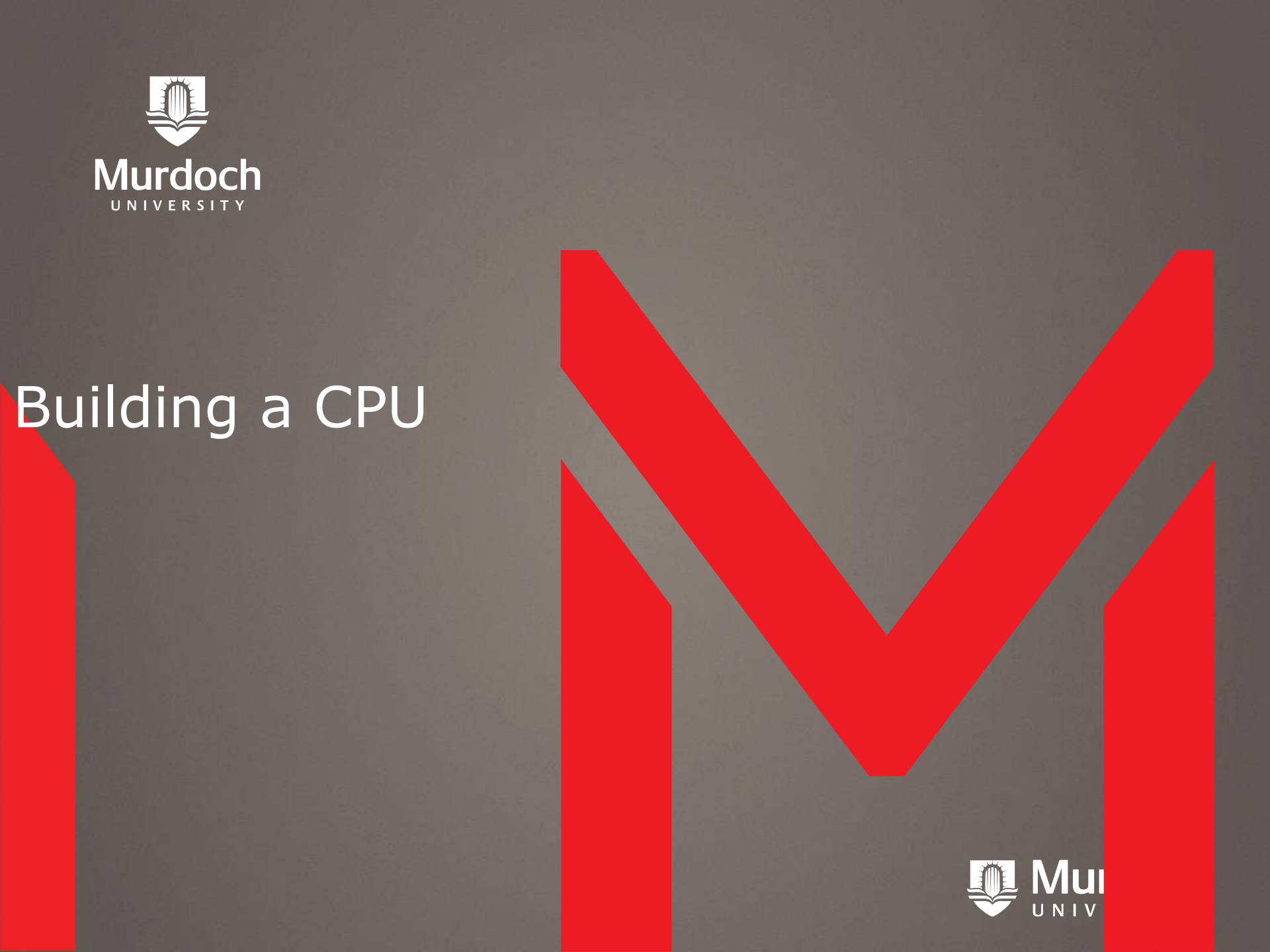
Reading

Title: **Structured Computer Organization (6th Edition)**
Author: Andrew S. Tanenbaum, Todd Austin,
Publisher: Prentice Hall
Keywords: organization, computer, structured
Pages: 800
Published: 2012-03-05
Language: English
Category: Design & Architecture, Hardware, Computers & Technology,
ISBN-10: 0132916525 ISBN-13: 9780132916523
Binding: Hardcover (6)
Reading: Chapter 3 "The Digital Logic Level"

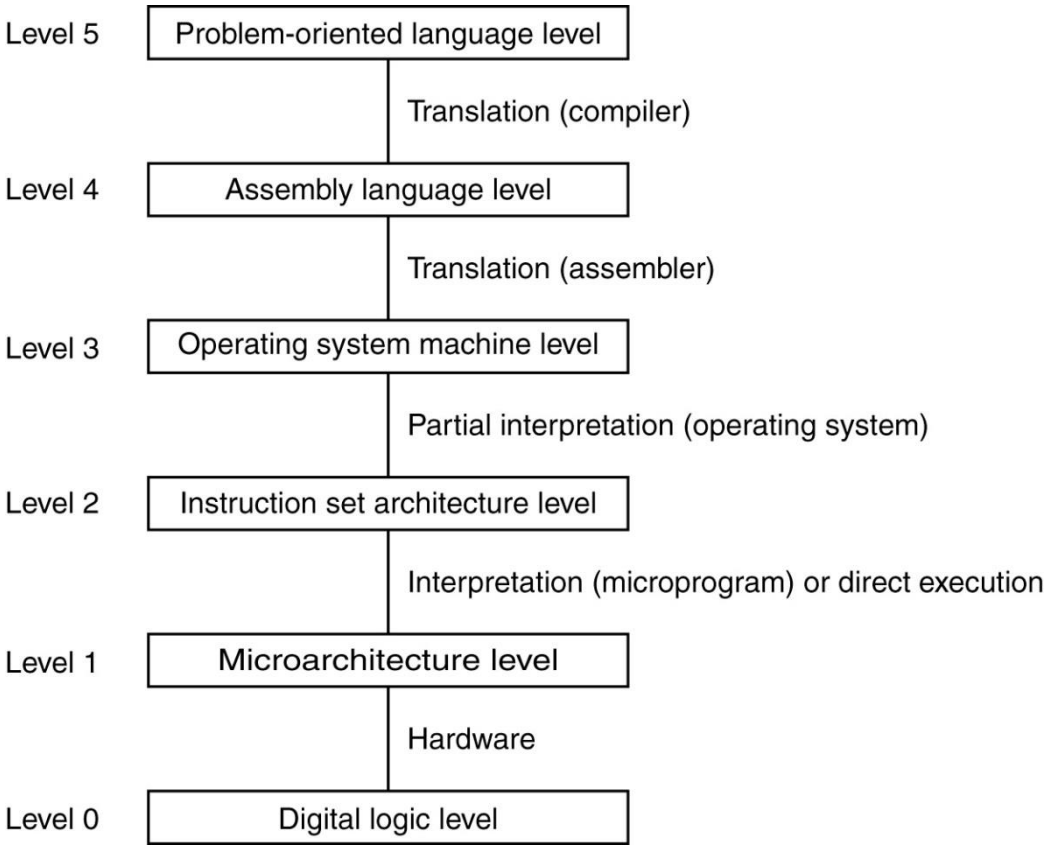


Resources:

- The recorded lectures available on LMS.
- The lecture slides available on LMS.


Murdoch
UNIVERSITY
Building a CPU
Mur
UNIV

Contemporary Multilevel Machines



A six-level computer

The support method for each level is indicated below it

Murdoch
UNIVERSITY

CPU

Mu
UNIV

The Central Processing Unit (CPU)

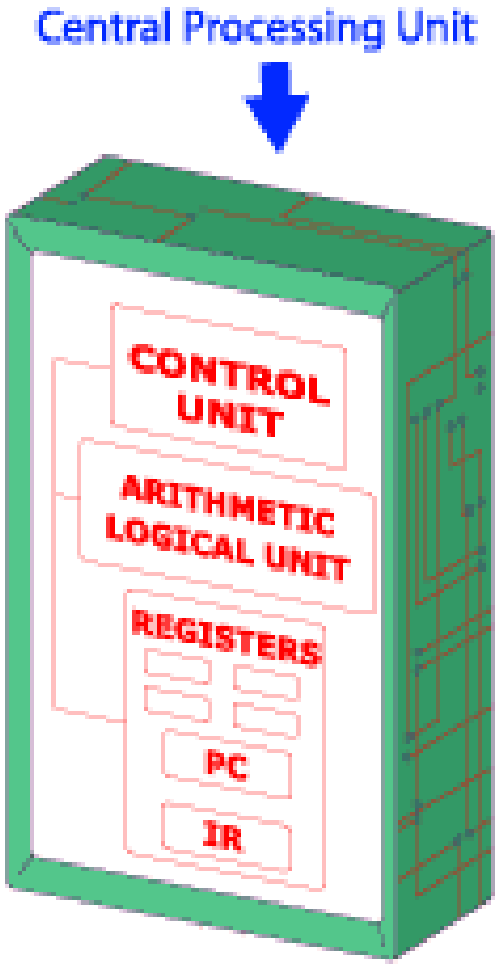
The **Central Processing Unit** is a silicon chip that is the 'brain' of a computer system.

It executes program instructions to control all the devices within the machine

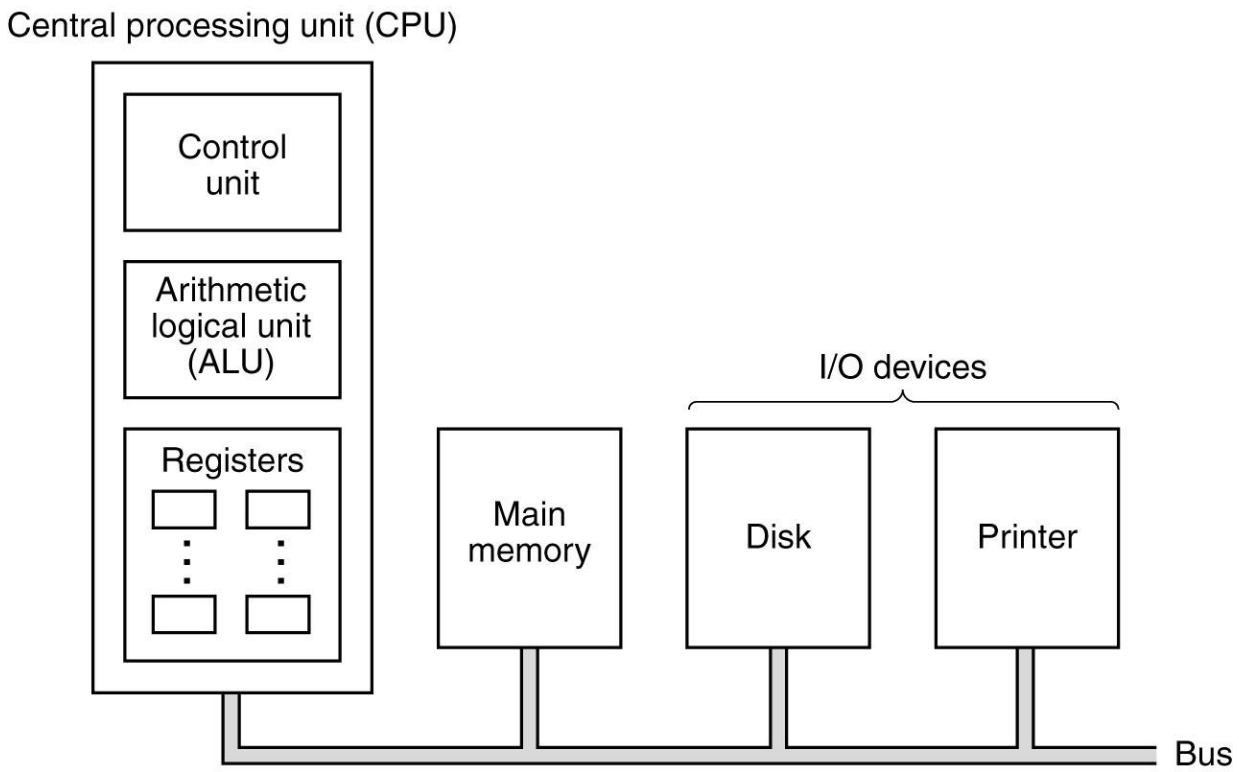
Its internal organisation (**architecture**) consists of 3 main parts:

- Control unit (CU)
- Arithmetic and Logic unit (ALU)
- Registers

Also called a Von Neumann Machine



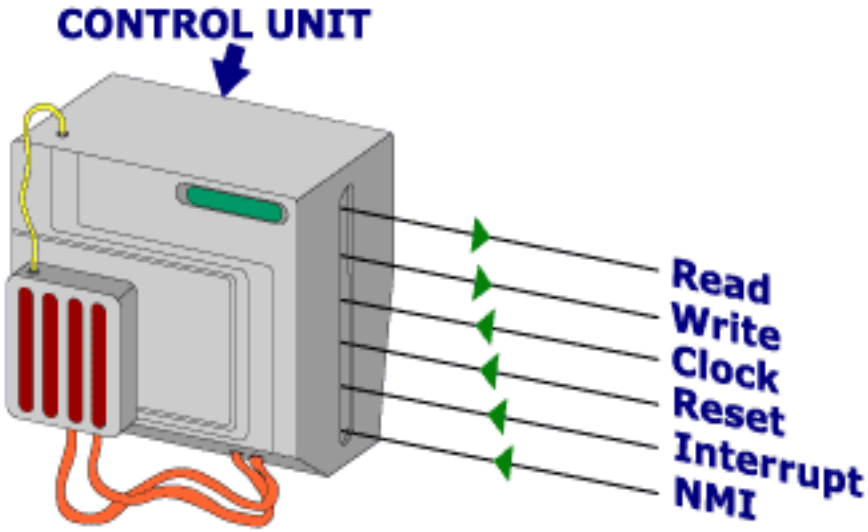
Central Processing Unit



The organization of a simple computer with one CPU and two I/O devices

The Control Unit

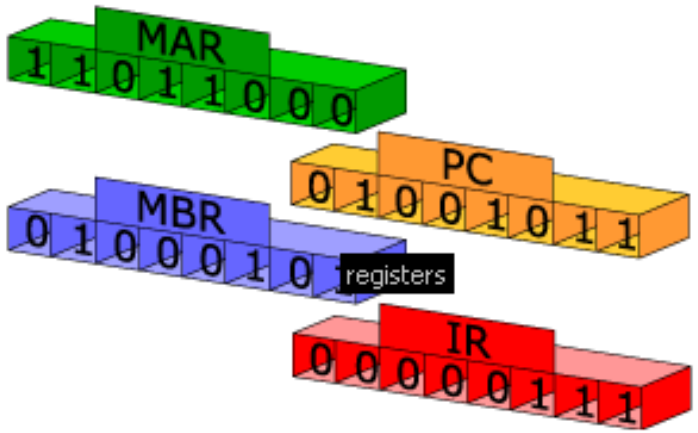
The **CU** sends signals to other parts of the computer



It manages fetch, decode and execute program instructions.

It **synchronises** the whole system by telling devices what to do and when to do it

The Registers



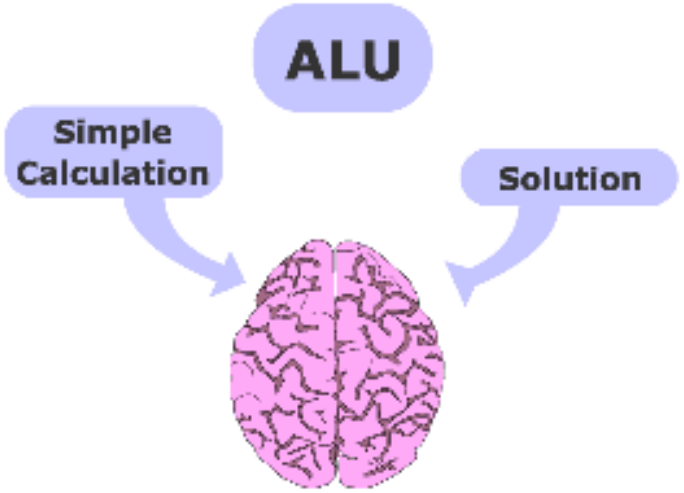
The **Registers** are very fast storage locations inside the processor itself. There are many registers including:

- **memory address register (MAR)** – holds the address of a location in memory
- **memory data register (MDR)** – holds data just read from or written to memory
- **program counter (PC)** – holds the address of the next instruction to be fetched
- **Instruction register (IR)** – holds the current instruction being executed
- **general purpose registers** – can be used by programmers

The Arithmetic and Logic Unit

The **ALU** is where data is actually processed in the CPU

Processing data in the ALU involves doing **arithmetic** calculations e.g. add, subtract, multiply, divide etc.



It also involves **logical** comparisons like AND, OR etc. using electronic circuitry

The ALU uses special arithmetic registers to temporarily store data and results of calculations e.g. the **accumulator**

CPU Organization: Instruction Types

- Two broad types:
  - Register-Memory
    - Allows Memory words to be fetched into registers
    - Allows registers to be stored back into memory
  - Register-Register
    - Fetches 2 operands from registers.
    - Brings them to the ALU input registers
    - Performs some operations
    - Puts result back into registers

CPU Organization: Instruction Types

- Data Manipulation
  - Add, subtract
  - Increment, decrement
  - Multiply
  - Shift, rotate
  - Immediate operands
- Data Staging
  - Load/store data to/from memory
  - Register-to-register move
- Control
  - Conditional/unconditional branches in program flow
  - Subroutine call and return

Murdoch
UNIVERSITY

Instruction Execution

Mur
UNIV

Sequences of Instructions

- A Computer Program is just a sequence of instructions
- We've seen how a single instruction is executed.
- What about more than one, i.e. a sequence
- We write a list of instructions.

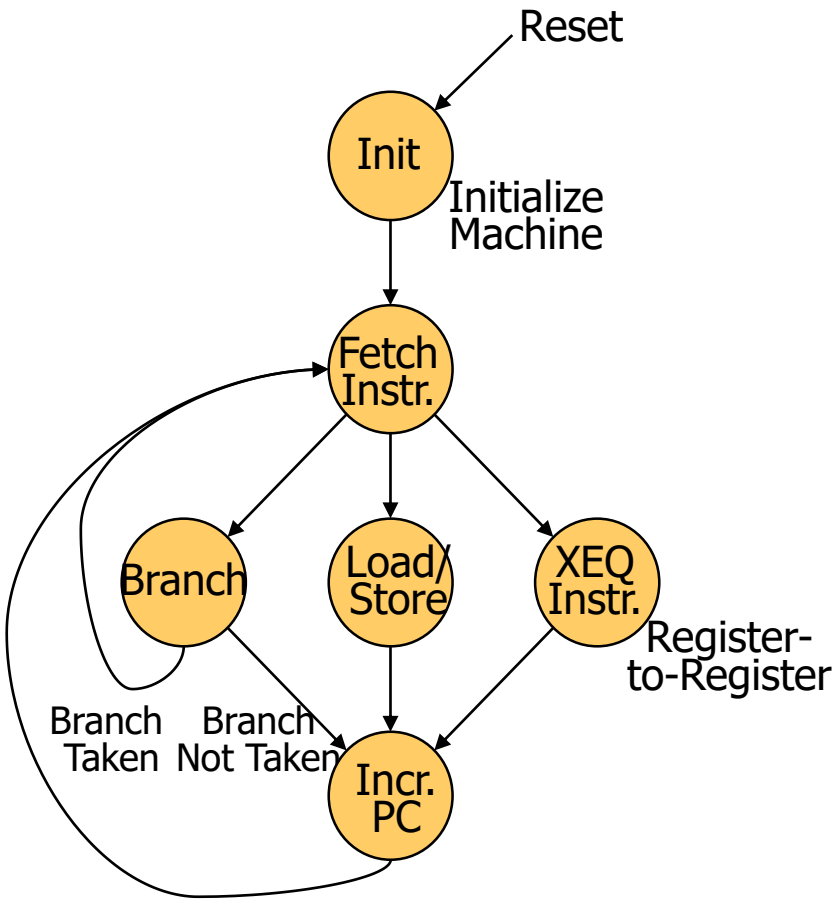
1. 3+3
2. 2-4
3. 4+5
4. 6+1

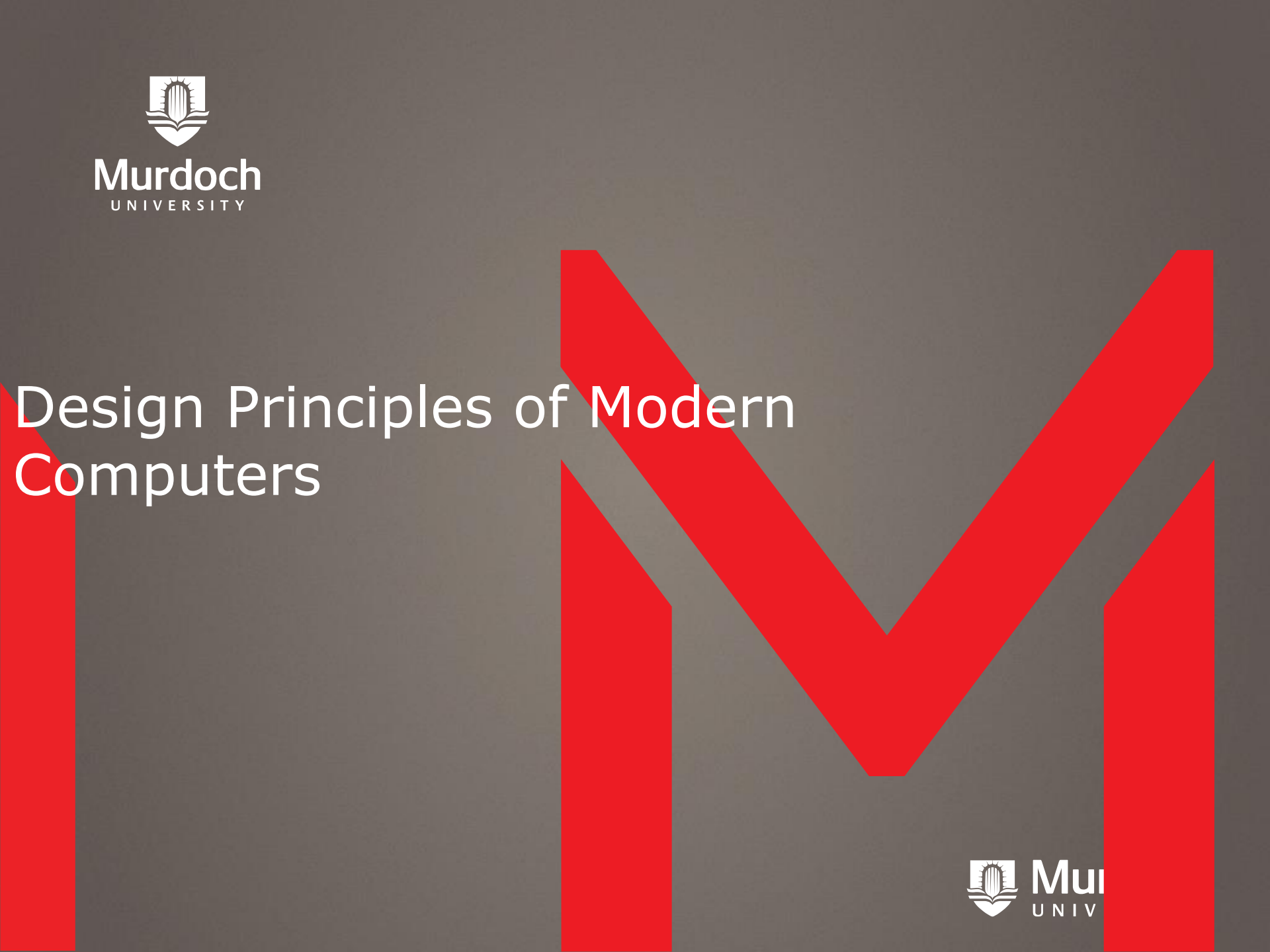
Instruction Execution Steps

1. Fetch next instruction from memory into instruction register
2. Change program counter to point to next instruction
3. Determine type of instruction just fetched
4. If instructions uses word in memory, determine where Fetch word, if needed, into CPU register
5. Execute the instruction
6. Go to step 1 to begin executing following instruction

Instruction Execution

- It's a little more complicated than that:
- Control State Diagram
  - Reset
  - Fetch instruction
  - Decode
  - Execute
- Instructions partitioned into three classes
  - Branch
  - Load/store
  - Register-to-register
- Different sequence through diagram for each instruction type




Murdoch
UNIVERSITY
Design Principles of Modern Computers
Mur
UNIV

Design Principles for Modern Computers

Rules that all Modern Computers aim to meet

- All instructions directly executed by hardware
- Maximise the rates at which instructions are issued
- Instructions should be easy to decode
- Only loads, stores should reference memory – minimal reference to memory

Design Principles for Modern Computers

Maximise rate at which instructions are issued

- One of many tricks to optimise execution
- Although instructions are encountered in program order, They don't always get executed in order, Instructions can be run out of order and in parallel
- So if you maximise issue rate, you may have massive performance gains
- This is one of the principles of parallel instruction execution

Design Principles for Modern Computers

- Instructions should be easy to decode
  - This is a critical limit in performance
  - So anything to improve this will help
    - Regular instructions
    - Fixed lengths
    - Small number of fields
    - The less complicated – the better

Design Principles for Modern Computers

- Only loads, stores should reference memory
  - Memory access takes a long time in comparison to register access
  - loads and store instructions are the only instructions to access slow memory
    - They can be run in parallel to faster instructions
  - We are building up the ability to parallelise instructions

Design Principles for Modern Computers

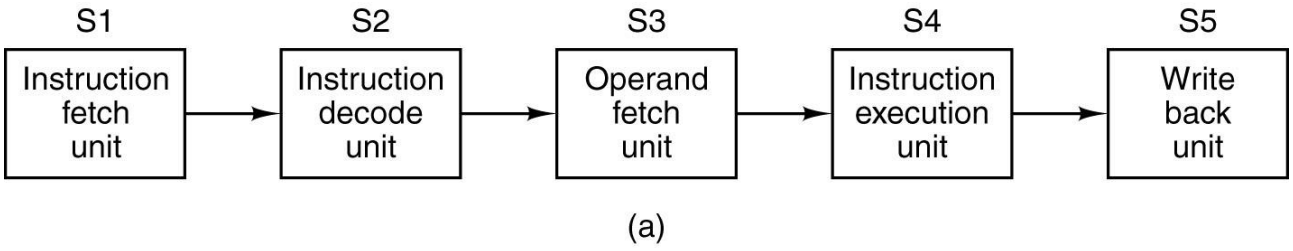
- Provide plenty of registers
  - Memory access is slow
  - Transferring from memory to registers takes time
  - Do as much as possible in the registers!

Design Principles for Modern Computers

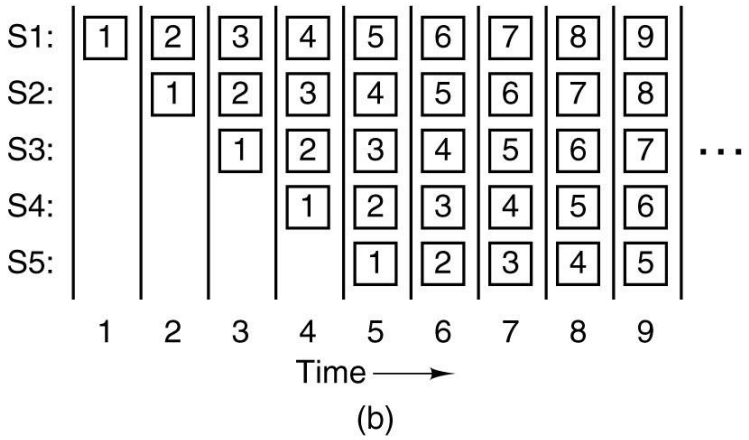
- On this basis all Architectural improvements and tweaks are based on.
  - Instruction-level Parallelism
  - Superscaler Architectures
  - Processor-Level Parallelism
  - Multiprocessors


Murdoch
UNIVERSITY
Processor Parallelism

Instruction-Level Parallelism



(a)



(b)

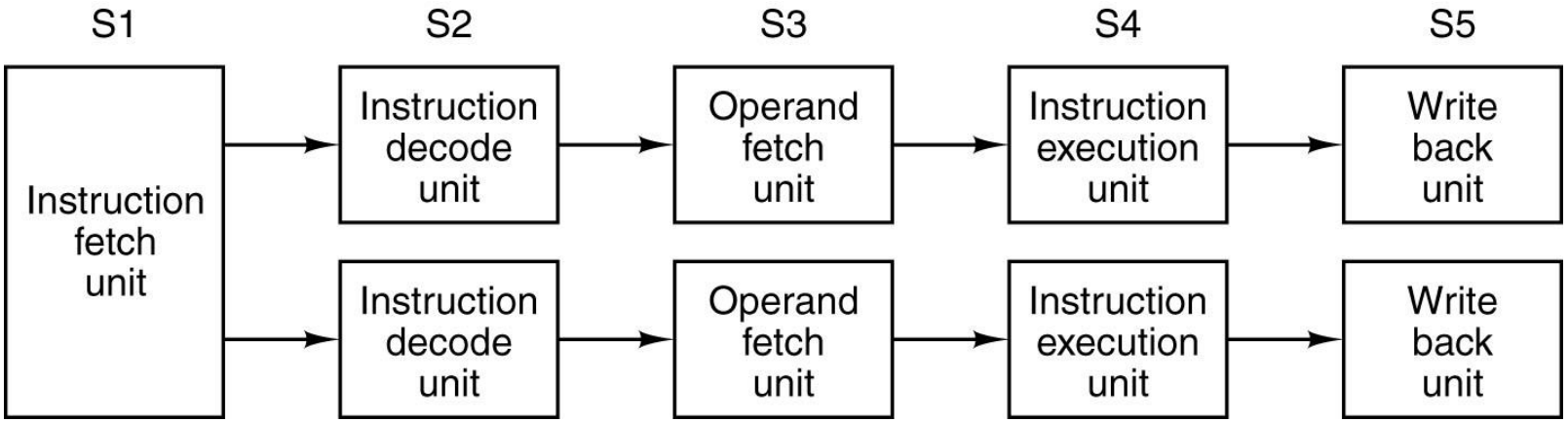
Sx = stage

Numbers are Program ID

A five-stage pipeline

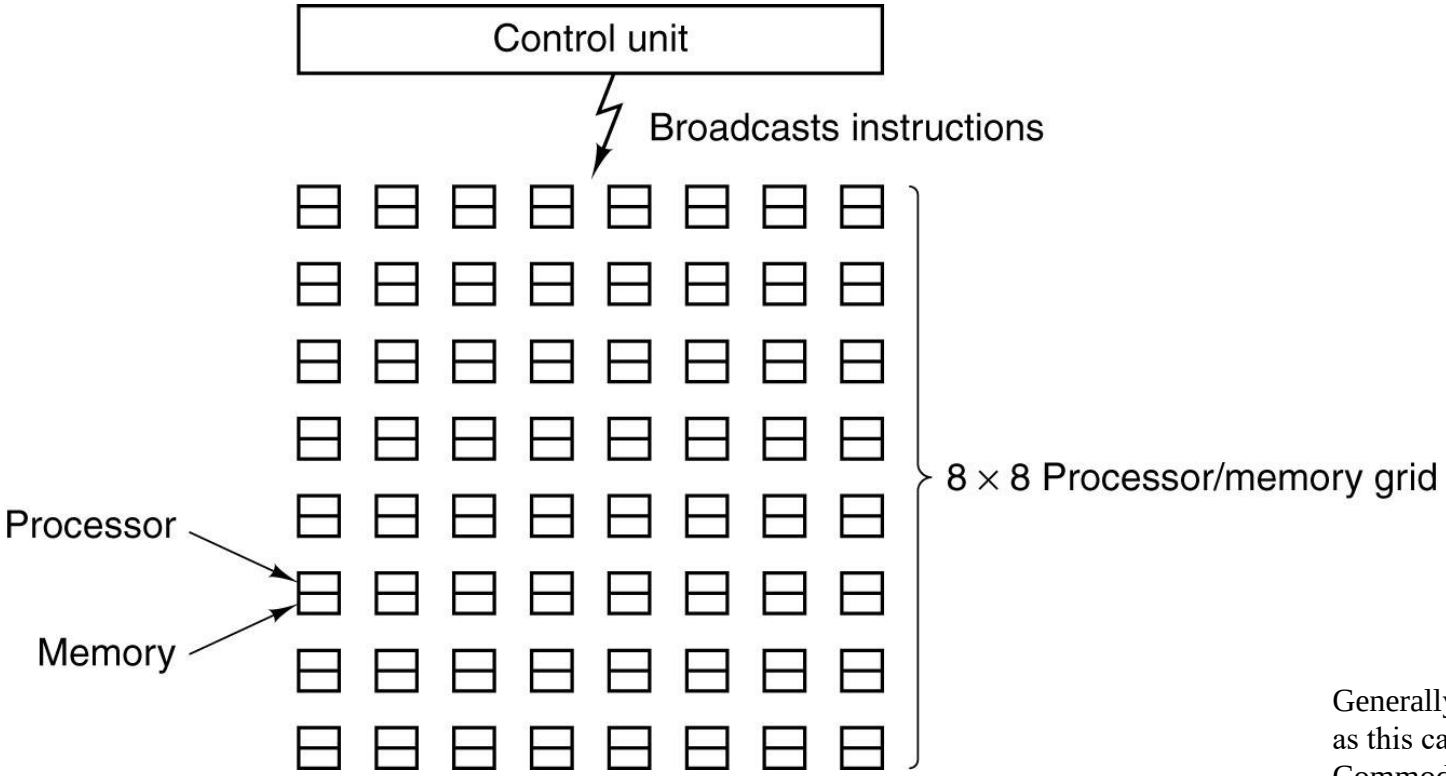
The state of each stage as a function of time. Nine clock cycles are illustrated

Superscalar Architectures (1)



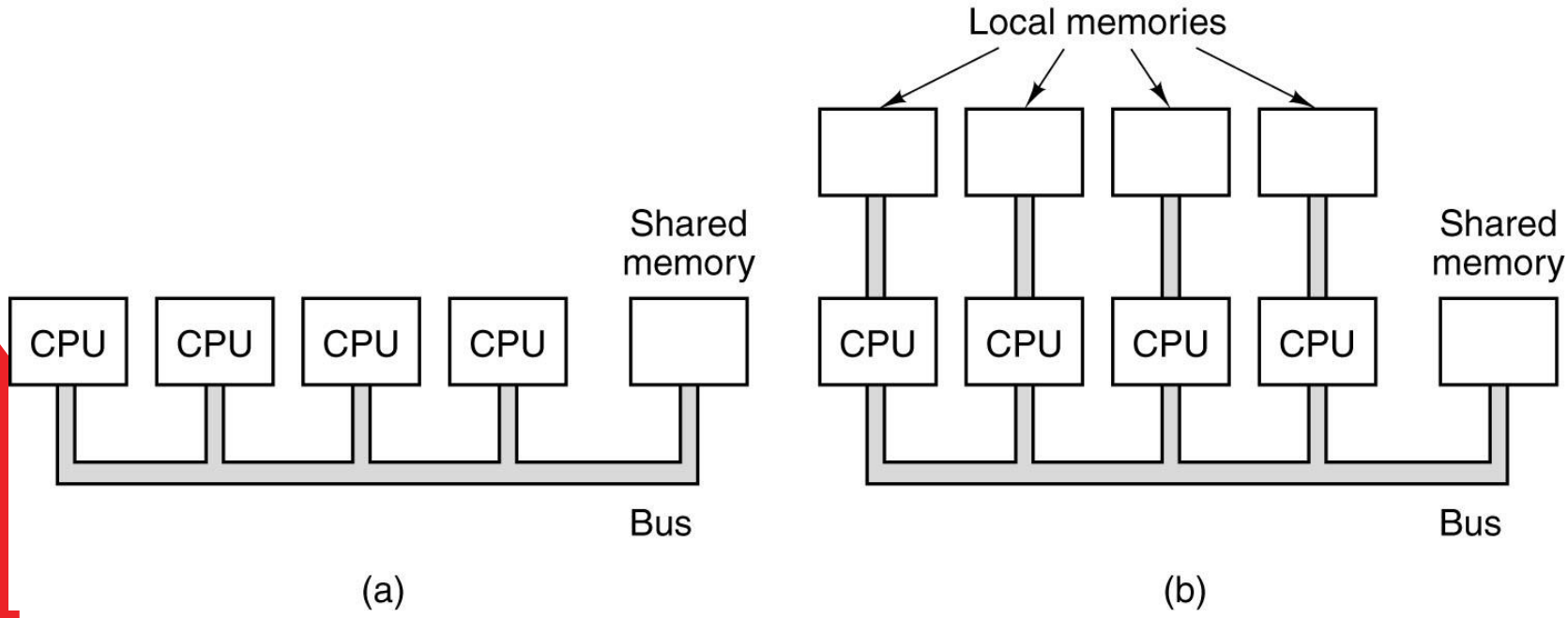
Dual five-stage pipelines with a common instruction fetch unit.

Processor-Level Parallelism (1)



Generally not used
as this can't be built from
Commodity CPUs

Processor-Level Parallelism (2)



A single-bus multiprocessor.

A multicomputer with local memories.

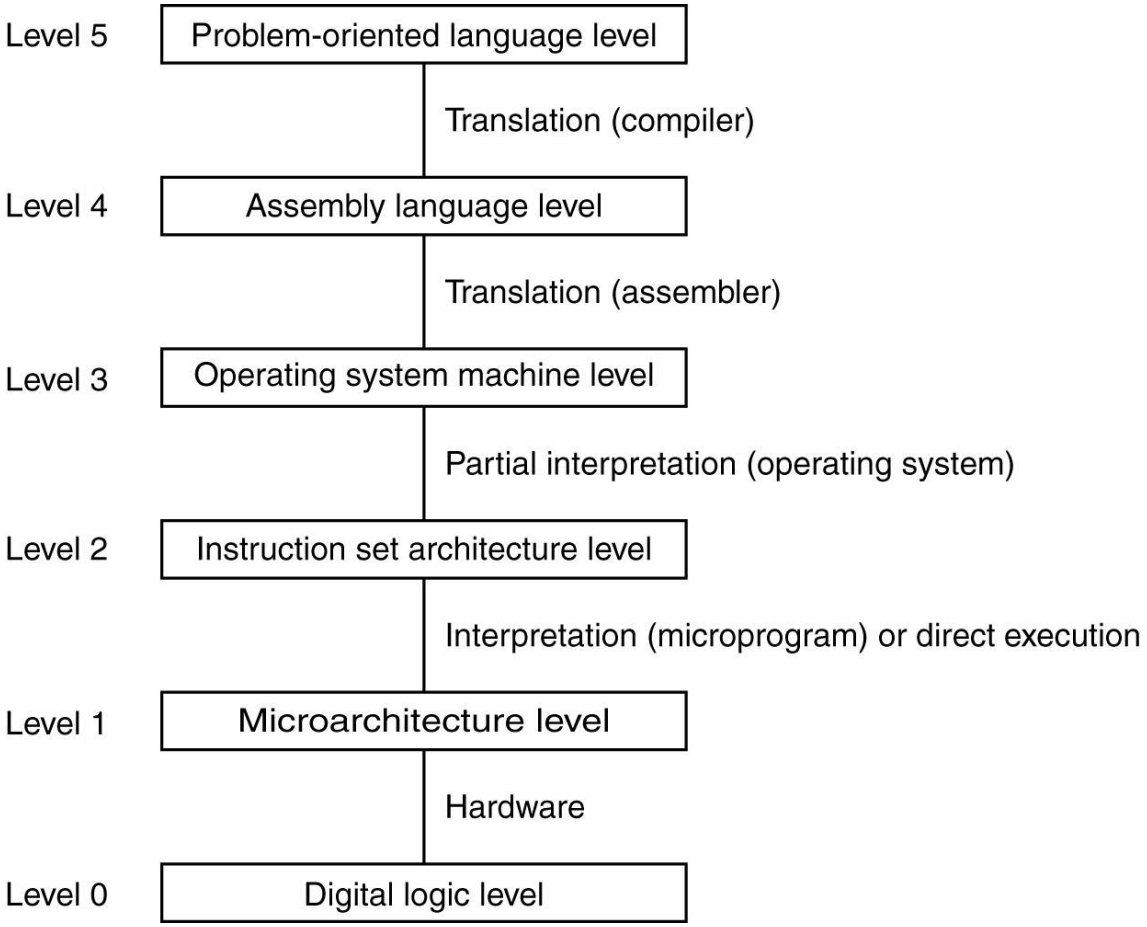
All modern CPUs are like this!

Murdoch
UNIVERSITY

The Microarchitecture Level

Mur
UNIV

Contemporary Multilevel Machines



A six-level computer

The support method for each level is indicated below it

The Microarchitecture Level

- Its job is to implement the Instruction Set of a processor on the underlying Digital Logic
- It takes higher-level instructions and figures out how to run them on the Digital Logic Circuitry
- Recall the Digital Logic description of a ALU!
  - It was a little dumb, just A+B, AB, `B and Adder.
  - To do more we need to build on these fundamental building blocks – this is what the microarchitecture level does!

The Microarchitecture Level: Characteristics

Goals depend on:

- The Instruction Set Architecture being implemented
- Cost and Performance Goals of the Computer

Many modern ISAs have instructions that can be implemented in a single clock cycle

- e.g. ARM architectures

More complex ISAs may require many cycles to execute a single instruction

- e.g. the Pentium 4

The Microarchitecture Level: Characteristics

Executing an Instruction Set Architecture instruction may require the microarchitecture to:

- Locate operands in Memory
- Read them
- Store results from the operation into Memory

Deals with the sequencing of operations within a single instruction.

- This can be different depending on the operation, the ISA and the goals of the computer

The Microarchitecture

- A small program that sits in a ROM of a CPU
- Performs what is called a fetch-execute cycle
  - It fetches the instruction and all that is needed from memory
  - It executes the instruction as a series of digital logic operations
- For example, ISA operations might include:
  - Opcode param
  - IMUL, IADD, ISTORE; to multiple, add and store ints.
  - The microarchitecture executes these instructions

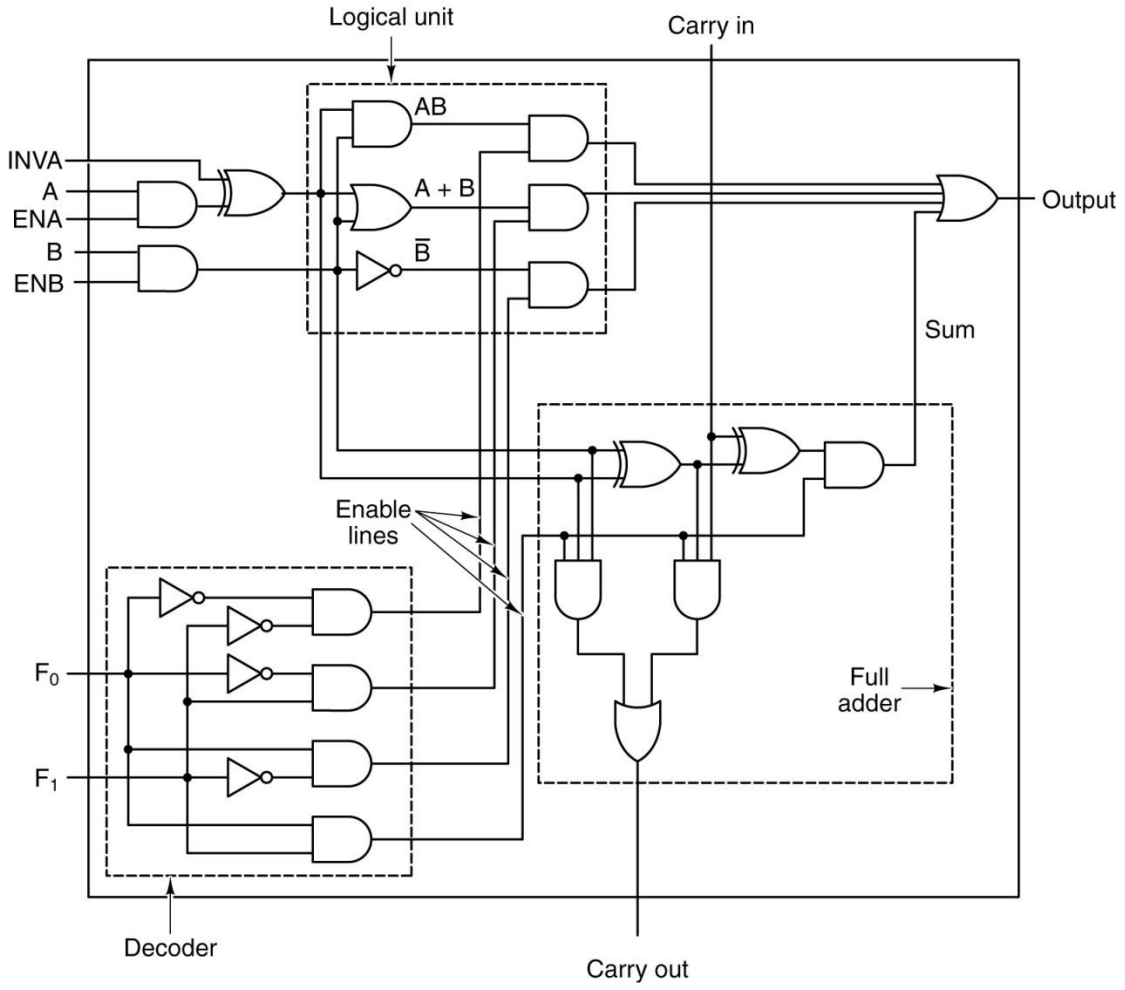
Recall: Arithmetic Logic Units (1)

Can do:
A AND B
A OR B
Inv B
A + B (full Adder)

Selected from:
F0, F1:
00, 01, 10, 11

ENVA ENVB:
Forces A or B to 0.
(Normally 1)

INVA: Inv A



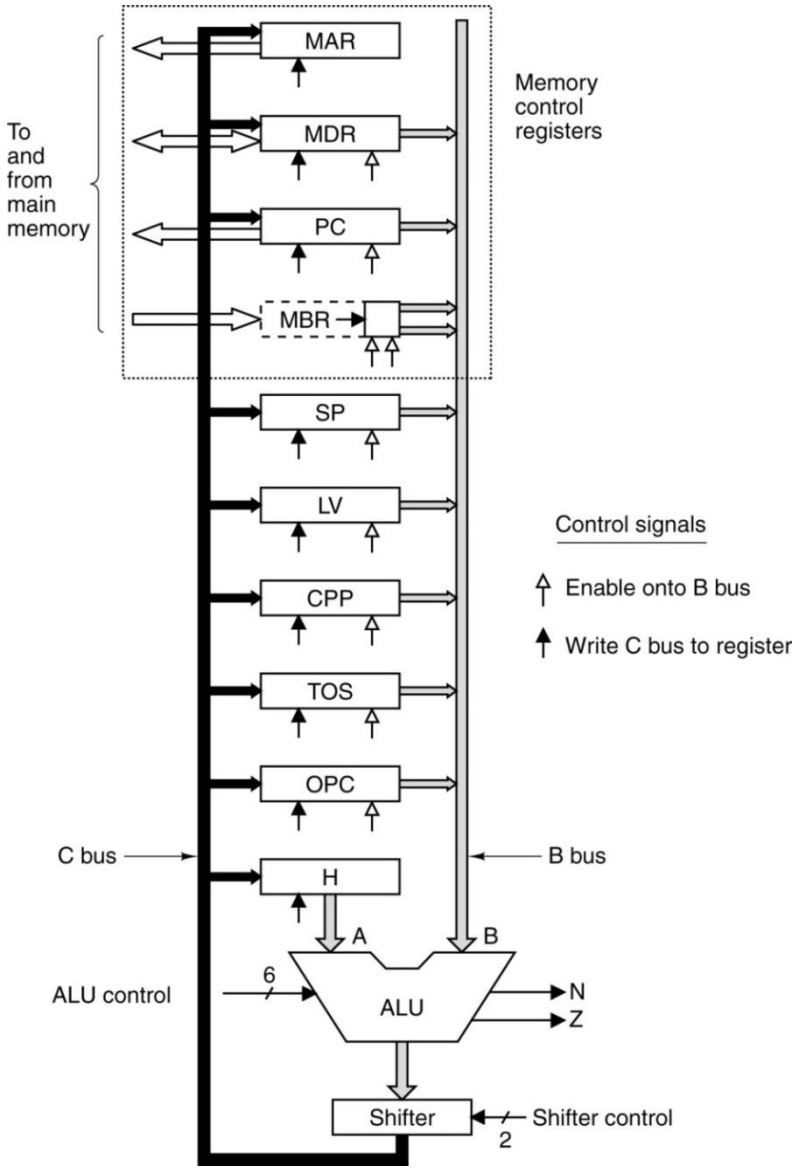
The Data Path (4)

- ALU Functions selectable through the control lines
- Recall how these are set using last weeks digital logic
- F0, F1, determines the ALU operation selection
- ENA,ENB – enable inputs A,B
- INVA – invert A
- INC – Increase by 1
- 2 other shift controls SLL, SRL
  - Shift Left/Right Logical

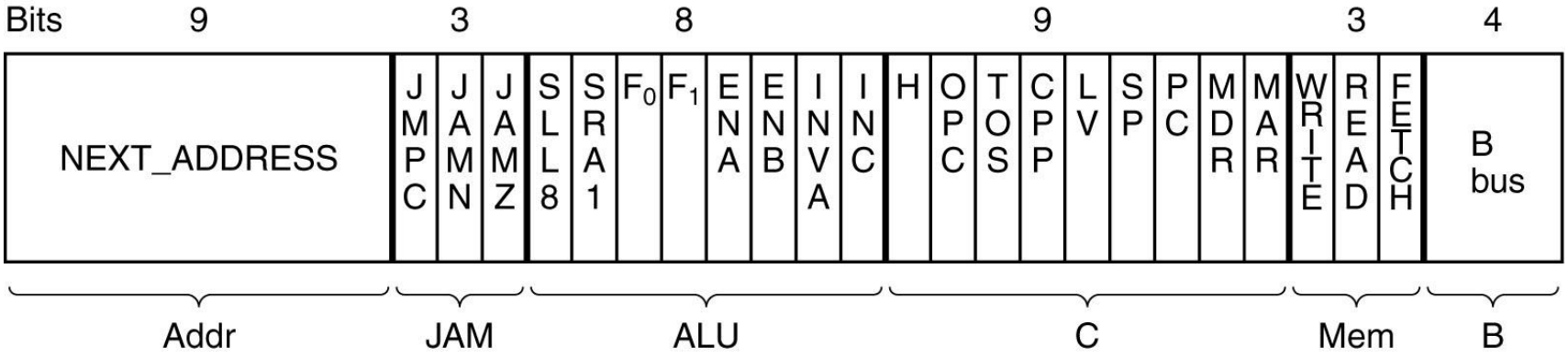
F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\overline{A}$
1	0	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1

Needs 29 Signals

- 9 to write from C bus
- 9 to write to B bus
- 8 to control ALU and shifters
- 2 to indicate r/w to MAR/MDR
- 1 for memory fetch via PC/MBR



Microinstructions



What does a microinstruction look like?

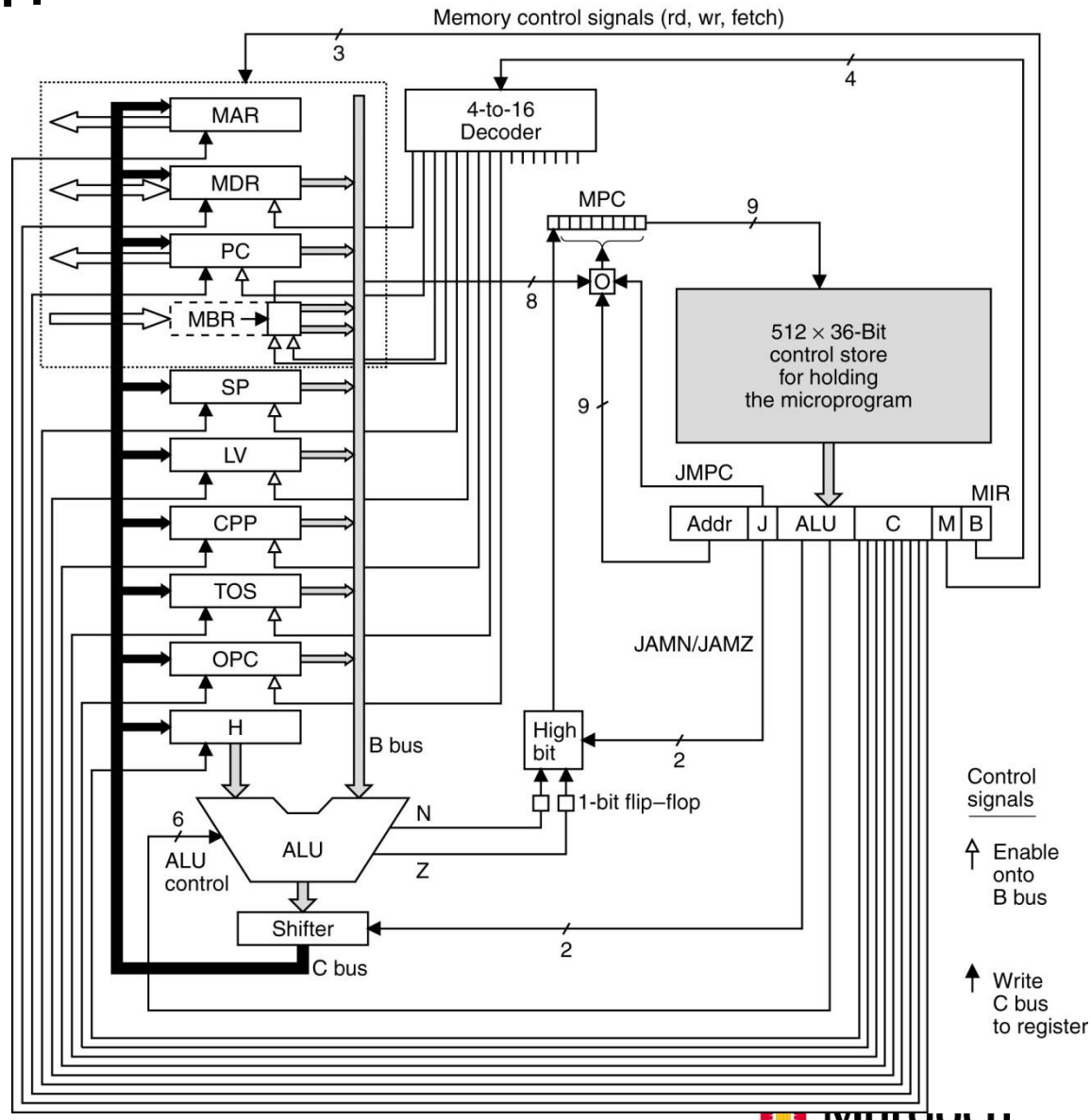
- Addr: address of a potential next microinstruction
- JAM: how next microinstruction is selected
- ALU: ALU and shifter functions
- C: which register written from C
- Mem: memory functions
- B: Source of B

B bus registers

0 = MDR	5 = LV
1 = PC	6 = CPP
2 = MBR	7 = TOS
3 = MBRU	8 = OPC
4 = SP	9-15 none

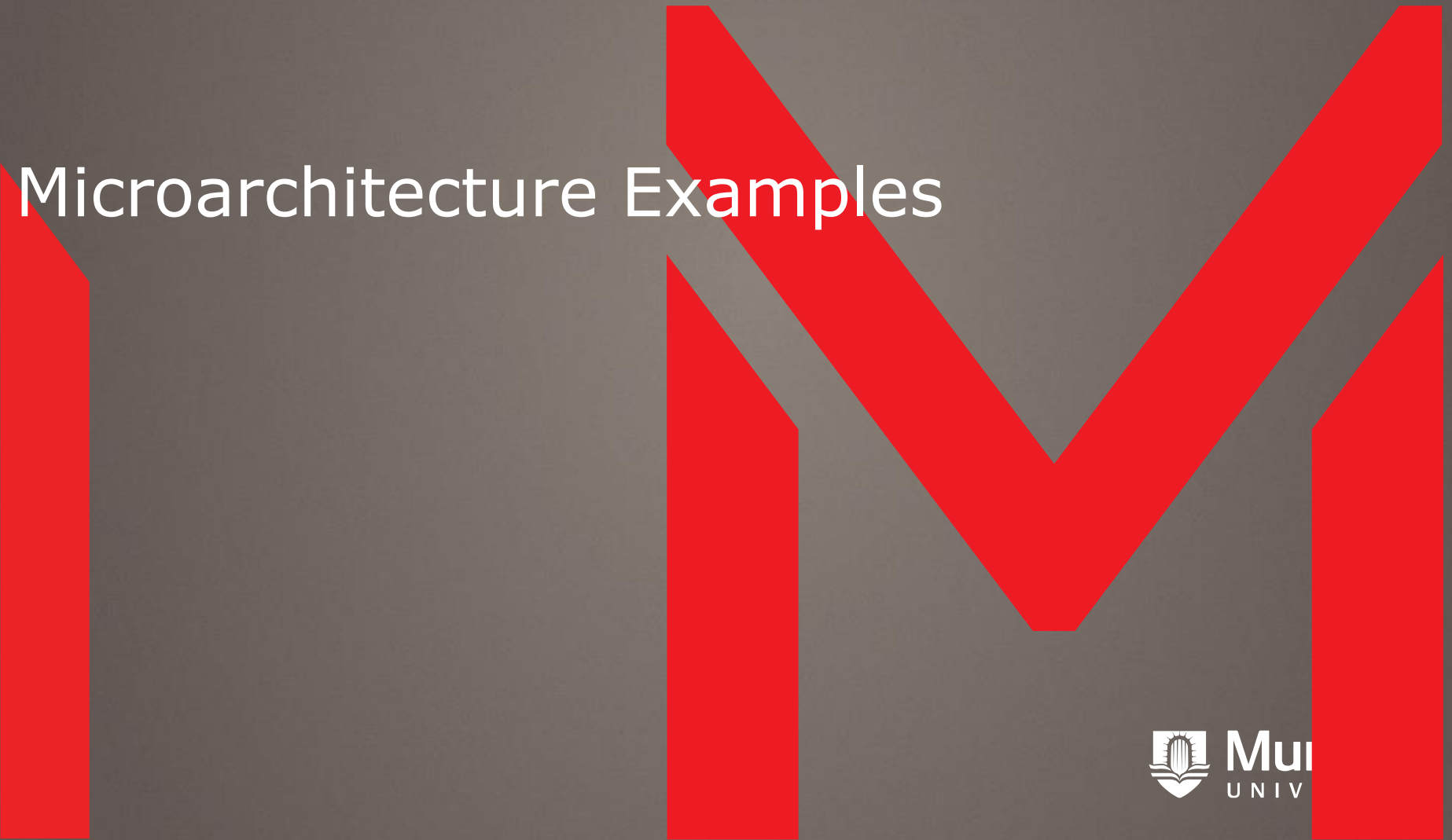
Microinstruction Control

So, this is how microinstruction instruction maps to the Physical Machine!



Murdoch
UNIVERSITY

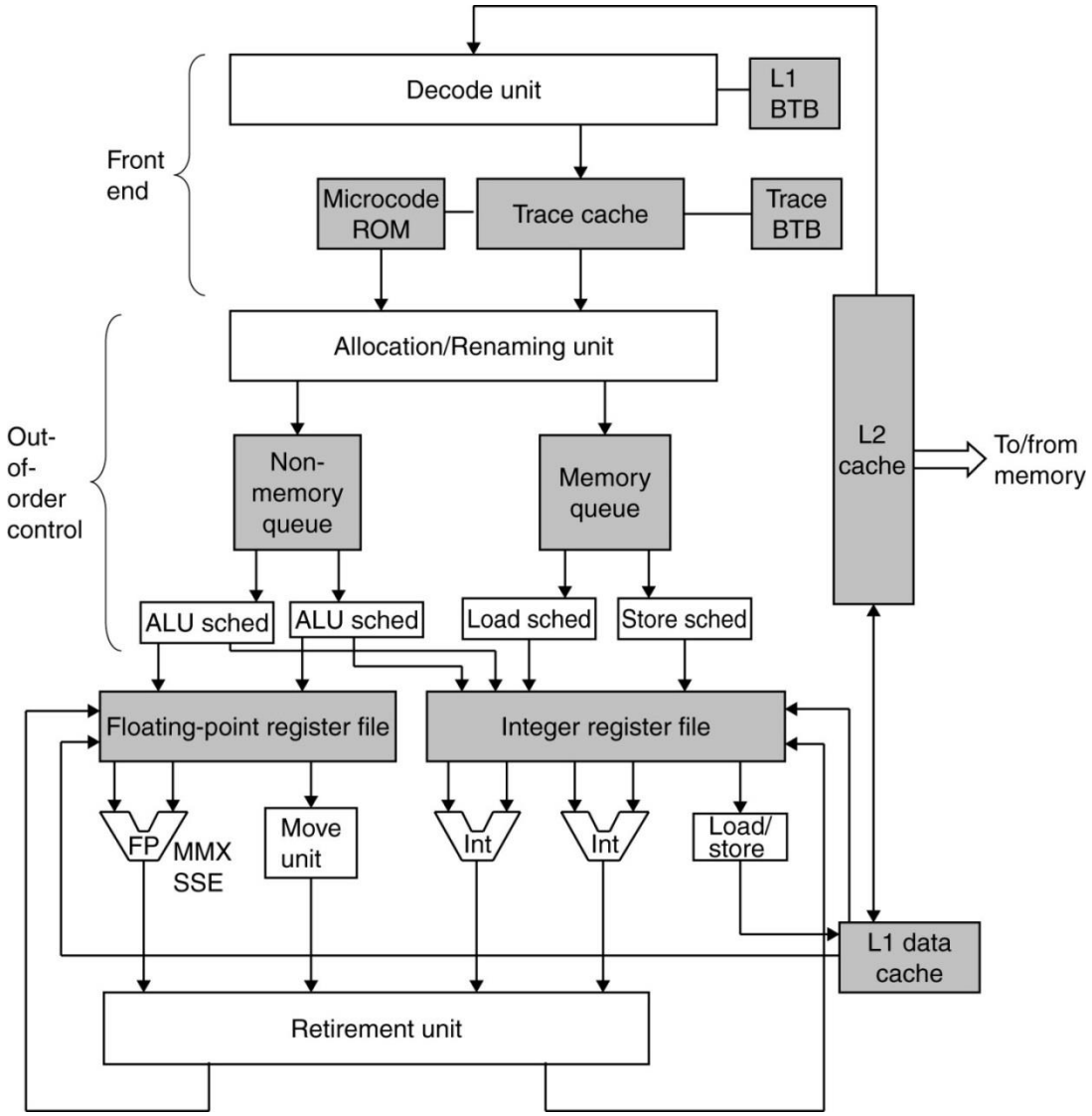
Microarchitecture Examples



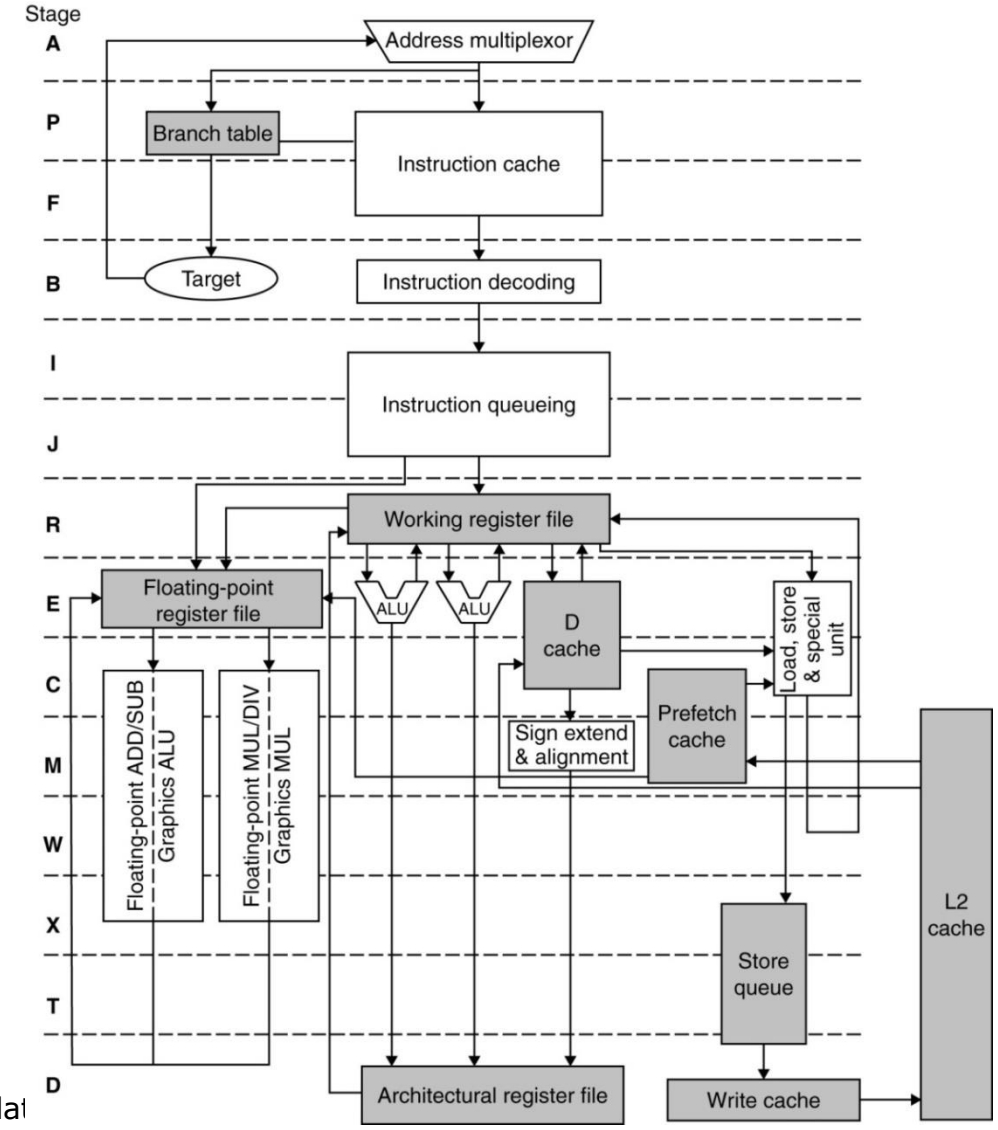
Mur
UNIV

The NetBurst Pipeline

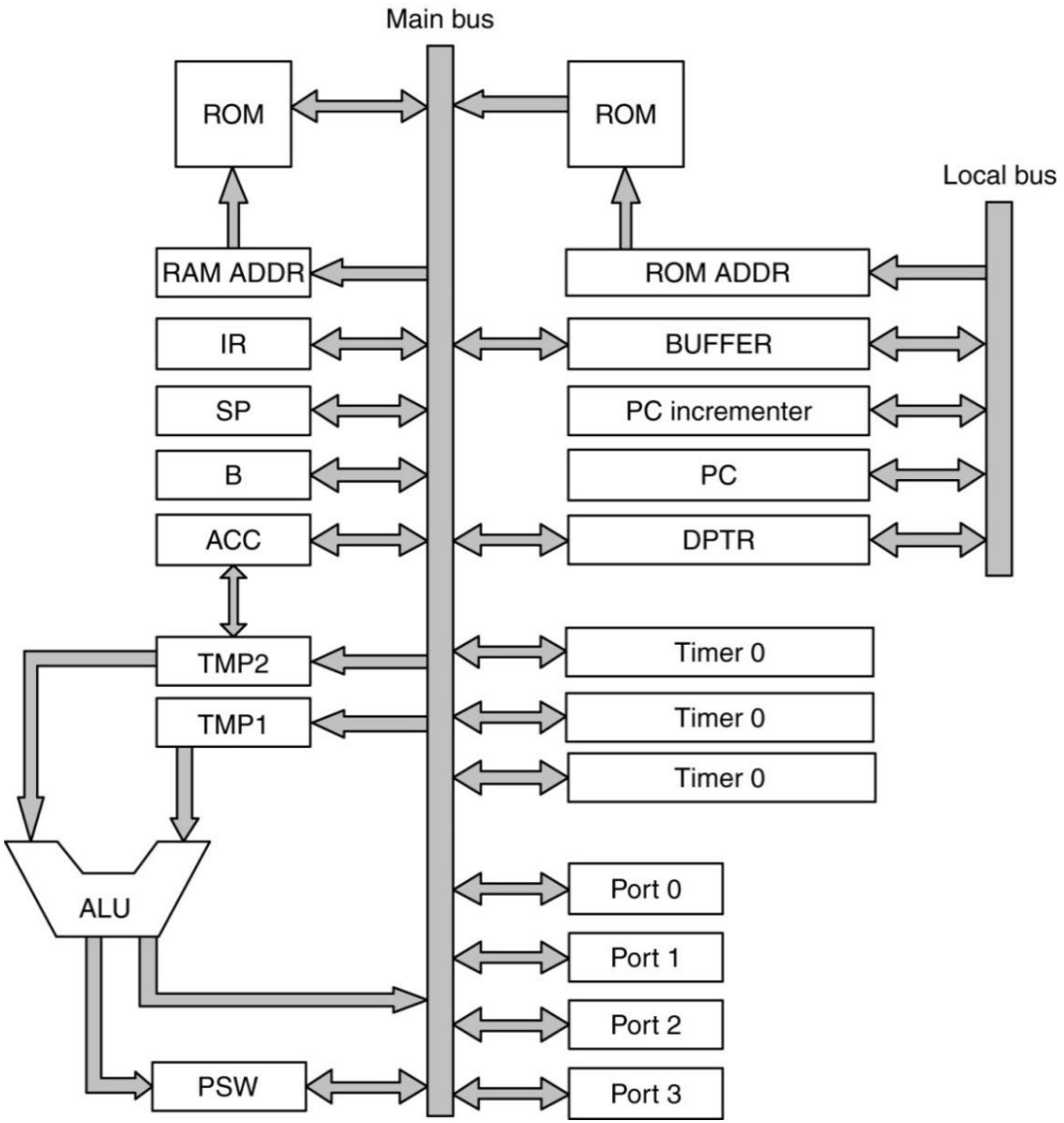
A simplified view of the Pentium 4 data path.



The UltraSPARC III Cu Microarchitecture



The Microarchitecture of the 8051 CPU

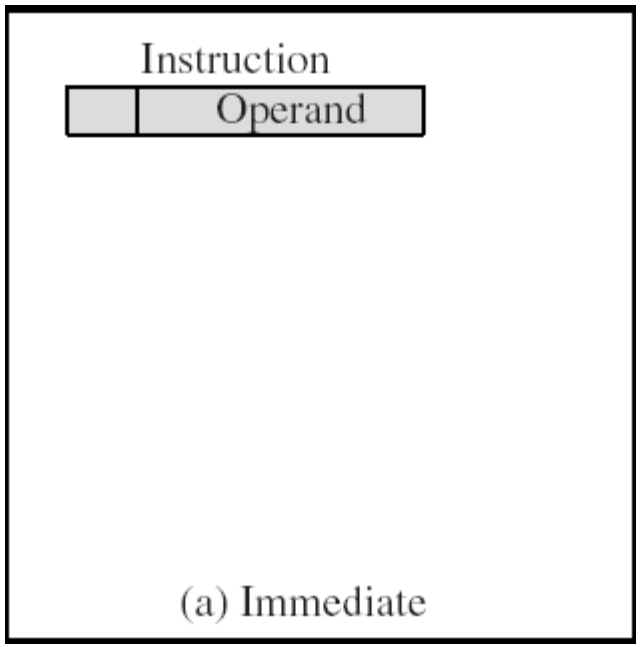



Murdoch
UNIVERSITY
Addressing Modes
Mu
UNIV

Addressing Modes

- Immediate
- Direct
- Indirect
- Register
- Register Indirect
- Displacement (Indexed)
- Stack

Immediate Addressing

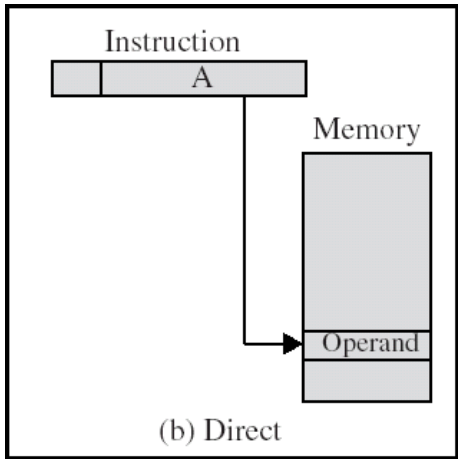


- Operand is part of instruction
- Operand = address field
- e.g. ADD AX, 0Ah
- LDA #0Ah

  Add 10 to contents of accumulator

  10 is operand
- No memory reference to fetch data
- Fast
- Limited range

Direct Addressing



- Address field contains address of operand
- Effective address EA = address field (A)
- ADD AX, value
- Value DB 0Ah

Add contents of cell value to accumulator AX

Look in memory at address value for operand

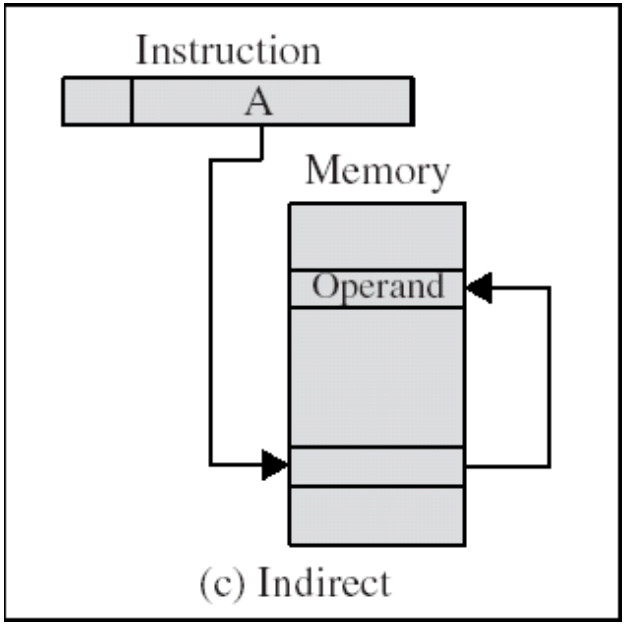
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space

Indirect Addressing

- Memory cell pointed to by address field contains the address of (pointer to) the operand
- EA =(A)

  Look in A, find address (A) and look there for operand
- e.g. ADD AX, (A)

  Add contents of cell pointed to by contents of A to accumulator



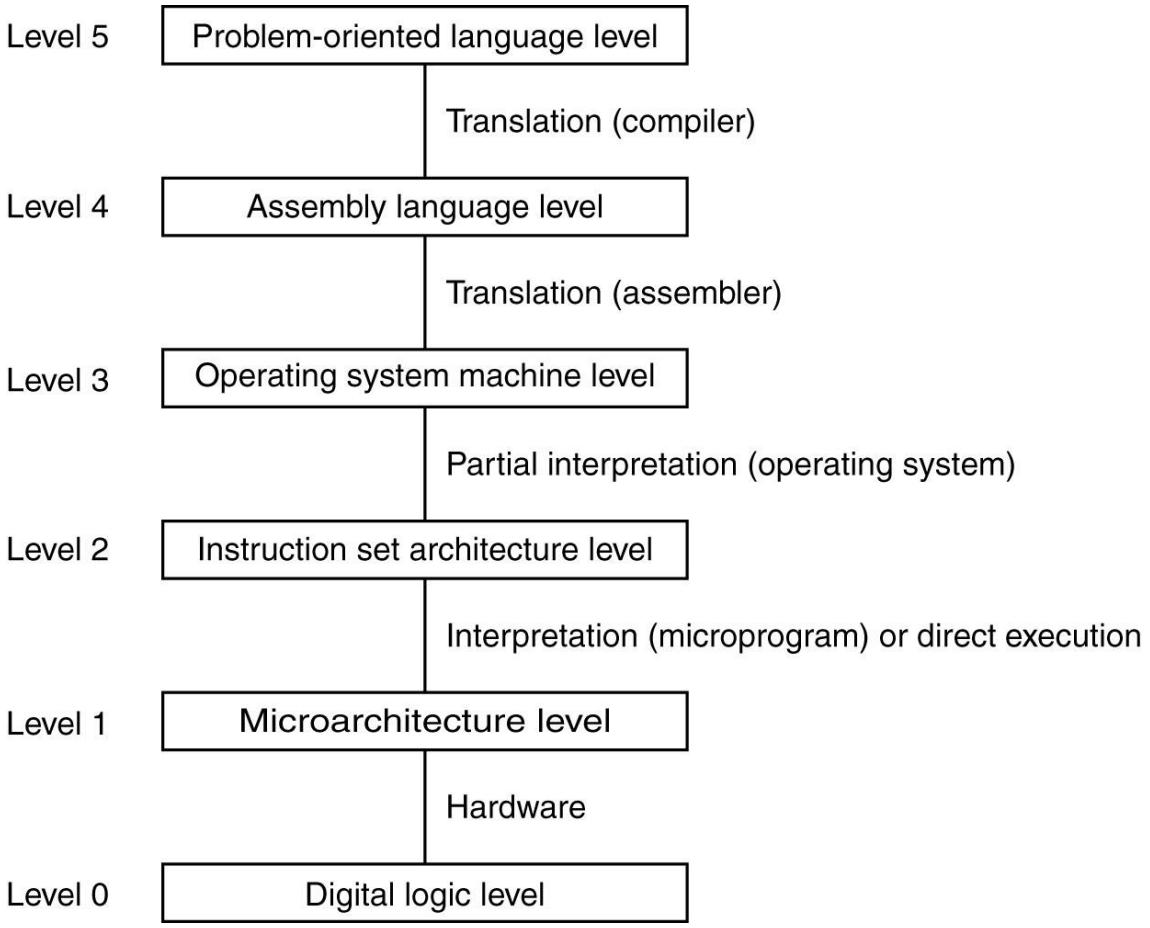
Murdoch
UNIVERSITY

The Instruction Set Architecture

Mur
UNIV

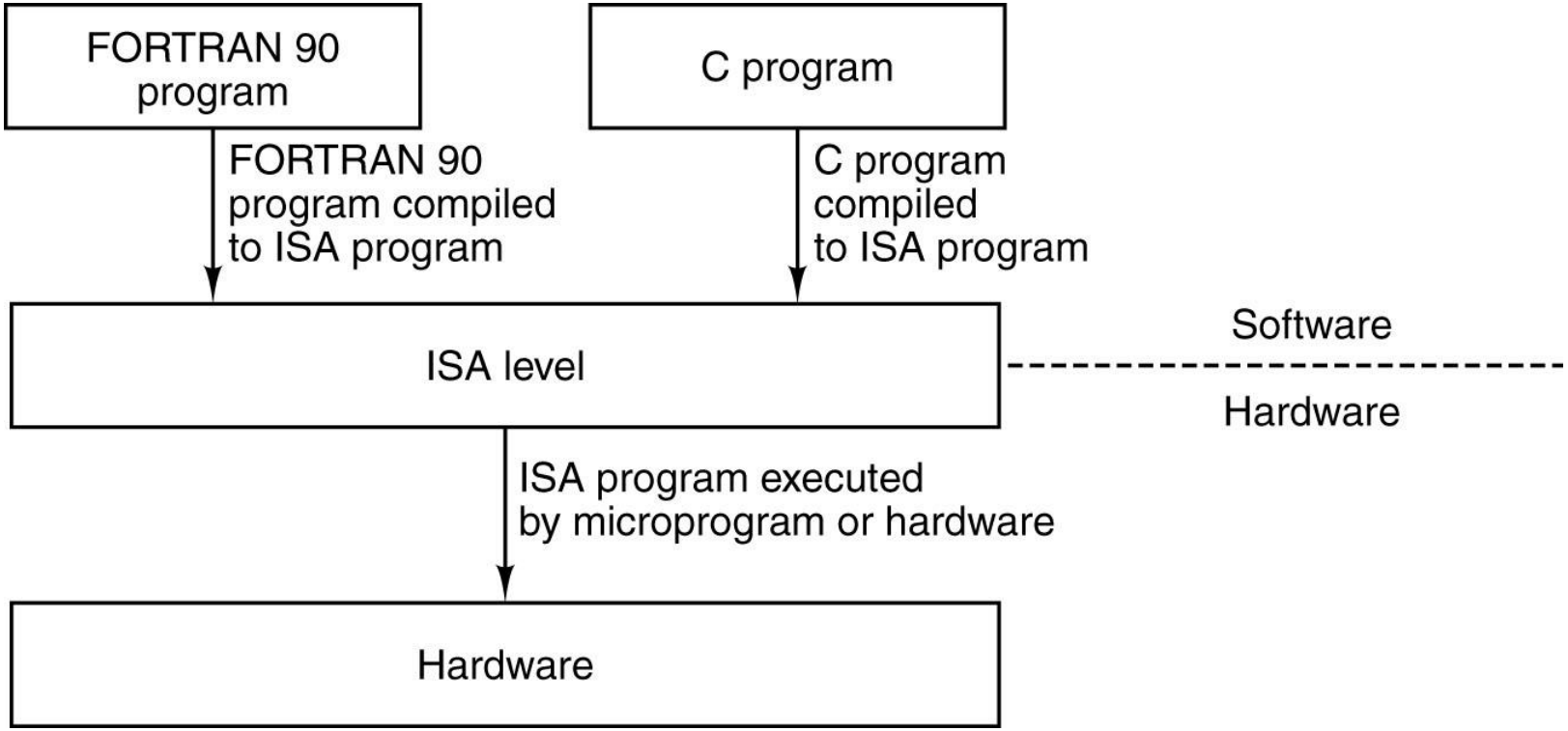
The Instruction Set Architecture level

The level above the Microarchitecture level of our 5 level computer



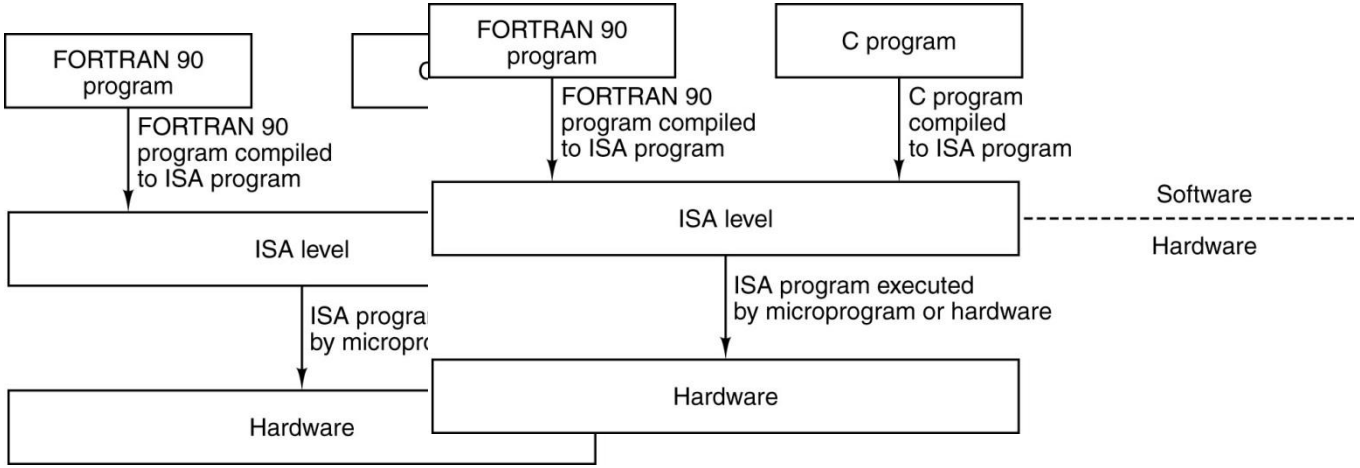
ISA Level

The ISA level is the interface between the compilers and the hardware.



ISA Level: The theory

- Instead of having the hardware execute C/Java/etc programs on the hardware directly – this would be very hard..
- Translate programs in high-level languages to a common intermediate form
- Programmers write to this ISA interface/level



Murdoch
UNIVERSITY

Supporting the ISA Level

Mur
UNIV

Supporting the ISA level (1/2)

- Programmers don't actually write code in the instruction set architecture language
- Compilers 'compile' programs from a high-level language to the ISA level.
- The ISA level defines the boundary between the hardware and the compiler
  - It is the language that both the compiler and the hardware must understand
  - Implemented by the microarchitecture in hardware

Supporting the ISA level (2/2)

- When designing a new machine:
  - The hardware architects talk to the compiler designers and agree on what is needed in the ISA-level
  - Features needed by the compiler are added to the ISA
  - Features deemed too complex to implement are not added and instead left to the compiler to implement at a higher level
  - Unnecessary hardware features are left out.

Murdoch
UNIVERSITY

Designing the ISA Level

Mur
UNIV

What makes a good ISA? (1/2)

1. Define a set of instructions that can be implemented efficiently

- By current and future technologies
- Results in cost-effective design over several generations of CPU technology.
- Poor designs are more difficult to implement and may require more gates and more memory
- Poor design may run slower.
- A design that takes advantage of a current technology might not be the best for the future

What makes a good ISA? (1/2)

2. Should provide a clean target for compiled code
   - Regularity and completeness in the range of options
   - Contain obvious – not crazy – options.

Should make the hardware designers happy (easy to implement) and the software designers happy (easy to generate code for)

Designing the ISA Level (1)

Properties of the ISA Level

- Recognise that ISA-code is what a compiler outputs
- The compiler needs to understand microarchitecture level properties of the machine, including:
- What the memory model is.
- What registers are available.
- What Data types and instructions are available

- On most machines, the ISA level defines Kernel and User modes

Designing the ISA Level (2)

- Not Properties of the ISA Level

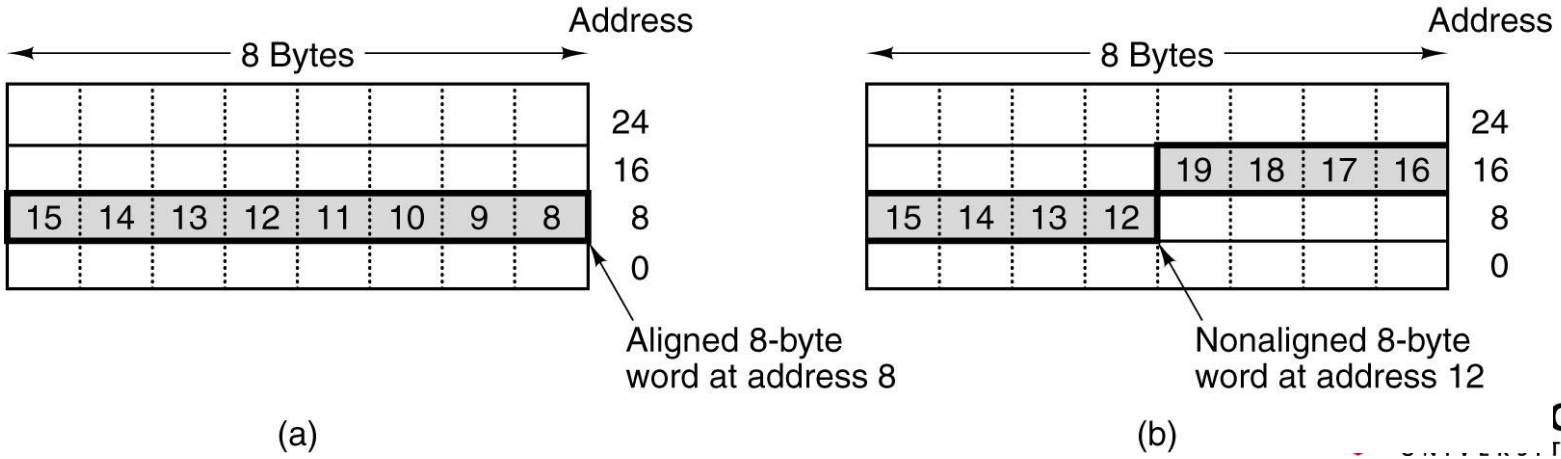
- Other issues are not part of the ISA as the compiler doesn't need to understand them:
- Hardware parallelism, superscaler design etc.
- The operation of the Control Unit and ALU.
- Optimisations at the CPU level e.g. converting Int to Float instructions

Designing the ISA Level (3)

Memory Model

- The compiler needs to understand how the computer organises its memory
- Most bytes are grouped into 4-byte(32-bit) or 8-byte (64-bit) words with instructions available for manipulating entire words.
- Compilers can use memory in whatever way they see fit.
- However, some machines require that words are aligned in memory cells, e.g.

An 8-byte word in a little-endian memory. (a) Aligned. (b) Not aligned. Some machines require that words in memory be aligned.



Designing the ISA Level (4)

Registers

- All computers have some registers visible and available to the ISA level/compiler
- In general registers at the microarchitecture level are not visible – e.g. Memory address registers
- Visible Register types:
  - General Purpose Registers
    - Hold local-variable, or intermediate results of calculations
    - Primary use is when you need maximum possible performance
    - Generally, providing more general purpose registers will improve potential program/compiler performance
  - Special Purpose Registers
    - For Program Counters, stack pointers etc.
    - Look at the instruction set for the Intel Pentium series of CPUs
    - Specialised instructions emerge every generation for e.g. graphics and complex maths

Designing the ISA Level (5)

Instructions

- Main feature of the ISA level is the machine instructions it contains
- Always will contain LOAD and STORE for moving data between memory and registers
- MOVE for copying data between registers
- Arithmetic instructions and boolean instructions
- Instructions for comparing data items and branching on the results
- Any Specialised instructions

Data Types at the ISA Level

- All Computers need to store data
- To store data efficiently, the CPU can provide hardware support for it.
- Data Types:
  - Numeric
  - Integers of multiple lengths, 8-bit, 16-bit, 32-bit, 64-bit
- Different size of words depending on size
  - Non-numeric
  - ASCII, UNICODE
  - CPU ISAs can have instructions to help managed non-numeric data-types

Data Types on the Pentium 4

Type	1 Bit	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Bit						
Signed integer		×	×	×		
Unsigned integer		×	×	×		
Binary coded decimal integer		×				
Floating point				×	×	

The Pentium 4 numeric data types.

Supported types are marked with ×.

Data Types on the UltraSPARC III

Type	1 Bit	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Bit						
Signed integer		×	×	×	×	
Unsigned integer		×	×	×	×	
Binary coded decimal integer						
Floating point				×	×	×

The UltraSPARC III numeric data types.

Supported types are marked with ×.

Data Types on the 8051

Type	1 Bit	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Bit	×					
Signed integer		×				
Unsigned integer						
Binary coded decimal integer						
Floating point						

The 8051 numeric data types.

Supported types are marked with ×.

Murdoch
UNIVERSITY

Instruction Types

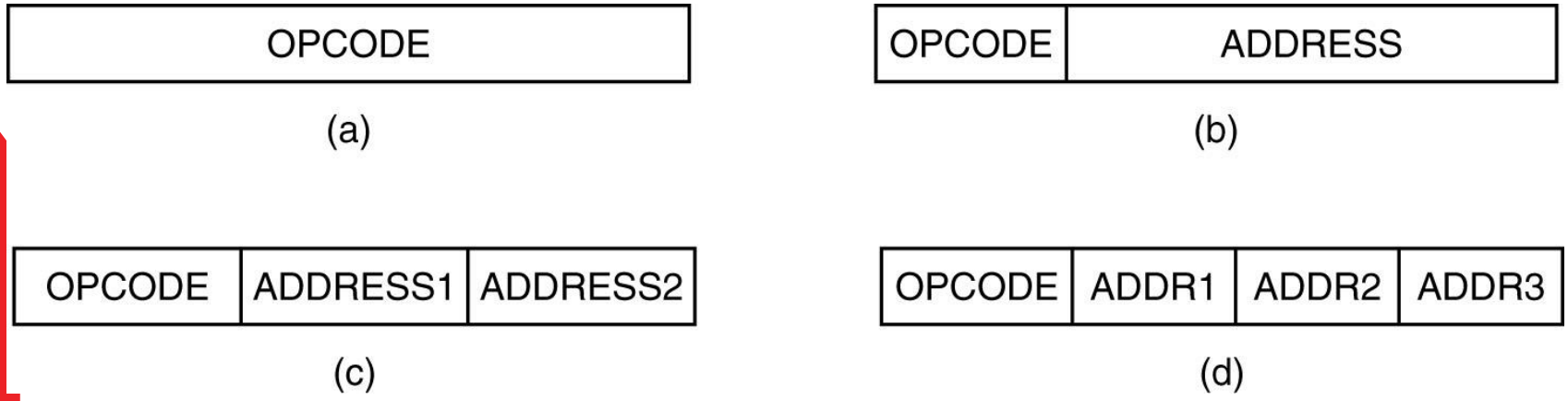
Mu
UNIV

Instruction formats at the ISA Level

- ISA Instructions consist of an opcode (operation code)
- Additional information such as where operands come from, where results go, etc
- Specifying where operands come from is called *addressing*
- Instructions can be the same length with padding
- Or variable size – requiring some sort of memory management

Instruction Formats

Example instructions:

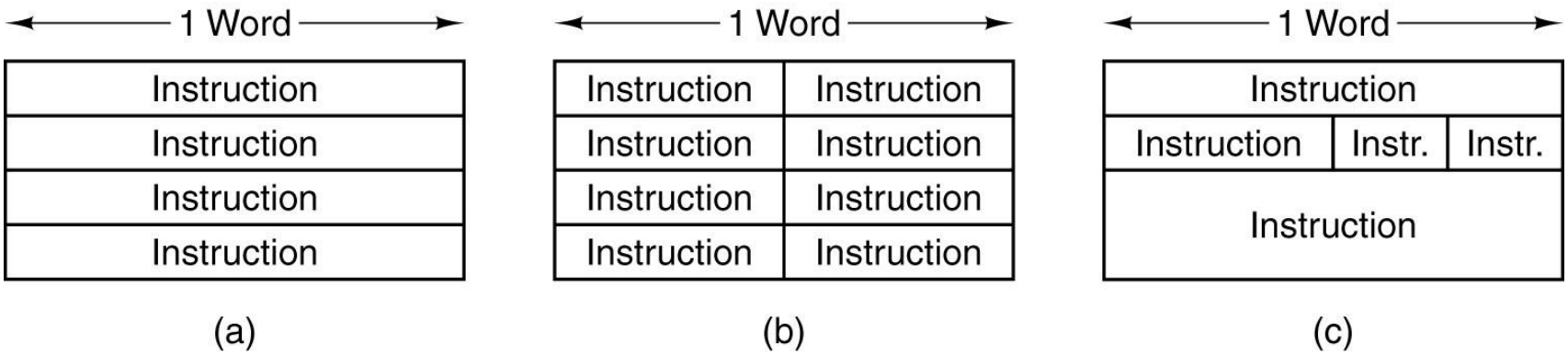


Four common instruction formats:

(a) Zero-address instruction. (b) One-address instruction

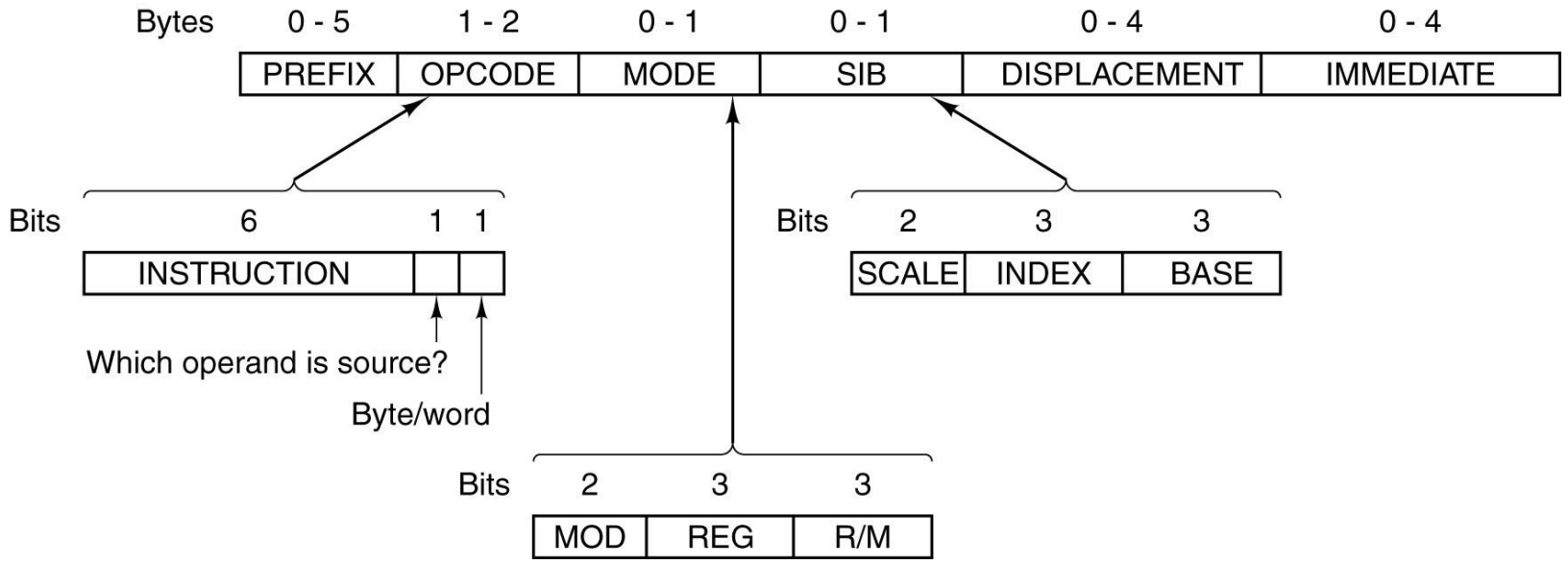
(c) Two-address instruction. (d) Three-address instruction.

Instruction Formats (2)



Some possible relationships between instruction and word length.

The Pentium 4 Instruction Formats



The Pentium 4 instruction formats.

Instruction types at the ISA Level

- There are many types of ISA instructions:
  - Data Movement Instructions
  - Dyadic Operations (combine 2 operands and produce a result)
  - Monadic (take one operand and produce one result)
  - Comparisons and Conditional Branches (test and branch control)
  - Procedure Call Instructions (A group of instructions that can be called)
  - Loop Control (Control repeated looping)
  - Input/Output (Control input and output)

Example: The Pentium 4 Instructions

A selection of the Pentium 4 integer instructions.

Moves

MOV DST,SRC	Move SRC to DST
PUSH SRC	Push SRC onto the stack
POP DST	Pop a word from the stack to DST
XCHG DS1,DS2	Exchange DS1 and DS2
LEA DST,SRC	Load effective addr of SRC into DST
CMOVcc DST,SRC	Conditional move

Arithmetic

ADD DST,SRC	Add SRC to DST
SUB DST,SRC	Subtract SRC from DST
MUL SRC	Multiply EAX by SRC (unsigned)
IMUL SRC	Multiply EAX by SRC (signed)
DIV SRC	Divide EDX:EAX by SRC (unsigned)
IDIV SRC	Divide EDX:EAX by SRC (signed)
ADC DST,SRC	Add SRC to DST, then add carry bit
SBB DST,SRC	Subtract SRC & carry from DST
INC DST	Add 1 to DST
DEC DST	Subtract 1 from DST
NEG DST	Negate DST (subtract it from 0)

A selection of the Pentium 4 integer instructions.

Binary coded decimal

DAA	Decimal adjust
DAS	Decimal adjust for subtraction
AAA	ASCII adjust for addition
AAS	ASCII adjust for subtraction
AAM	ASCII adjust for multiplication
AAD	ASCII adjust for division

Boolean

AND DST,SRC	Boolean AND SRC into DST
OR DST,SRC	Boolean OR SRC into DST
XOR DST,SRC	Boolean Exclusive OR SRC to DST
NOT DST	Replace DST with 1's complement

Shift/rotate

SAL/SAR DST,#	Shift DST left/right # bits
SHL/SHR DST,#	Logical shift DST left/right # bits
ROL/ROR DST,#	Rotate DST left/right # bits
RCL/RCR DST,#	Rotate DST through carry # bits

Example: The Pentium 4 Instructions

Test/compare

TEST SRC1,SRC2	Boolean AND operands, set flags
CMP SRC1,SRC2	Set flags based on SRC1 - SRC2

Transfer of control

JMP ADDR	Jump to ADDR
Jxx ADDR	Conditional jumps based on flags
CALL ADDR	Call procedure at ADDR
RET	Return from procedure
IRET	Return from interrupt
LOOPxx	Loop until condition met
INT n	Initiate a software interrupt
INTO	Interrupt if overflow bit is set

Strings

LODS	Load string
STOS	Store string
MOVS	Move string
CMPS	Compare two strings
SCAS	Scan Strings

Example: The Pentium 4 Instructions

Condition codes

STC	Set carry bit in EFLAGS register
CLC	Clear carry bit in EFLAGS register
CMC	Complement carry bit in EFLAGS
STD	Set direction bit in EFLAGS register
CLD	Clear direction bit in EFLAGS reg
STI	Set interrupt bit in EFLAGS register
CLI	Clear interrupt bit in EFLAGS reg
PUSHFD	Push EFLAGS register onto stack
POPFD	Pop EFLAGS register from stack
LAHF	Load AH from EFLAGS register
SAHF	Store AH in EFLAGS register

Miscellaneous

SWAP DST	Change endianness of DST
CWQ	Extend EAX to EDX:EAX for division
CWDE	Extend 16-bit number in AX to EAX
ENTER SIZE,LV	Create stack frame with SIZE bytes
LEAVE	Undo stack frame built by ENTER
NOP	No operation
HLT	Halt
IN AL,PORT	Input a byte from PORT to AL
OUT PORT,AL	Output a byte from AL to PORT
WAIT	Wait for an interrupt

SRC = source
DST = destination
# = shift/rotate count
LV = # locals

Murdoch
UNIVERSITY

Additional materials for the workshop (self studies)

Mur
UNIV

Machine codes

- Machine code or machine language is a set of instructions executed directly by a computer's central processing unit (CPU).

- Every processor or processor family has its own machine code instruction set.
- the instruction set is specific to a class of processors using (mostly) the same architecture.
- Successor processor designs often include all the instructions of a predecessor and may add additional instructions.
- Occasionally, a successor design will discontinue or alter the meaning of some instruction code (typically because it is needed for new purposes), affecting code compatibility to some extent

Machine codes – pros and cons

Advantages:

- Can have very specific instructions for processor features
- Allows the ongoing modification of the instruction set
- Allows for processors to have competitive advantage

Disadvantages:

- Almost impossible to program in
- Requires higher-level software to actually use
- Requires constant updates of system software
- Requires constant updates of development tools

Assembly Language - Why

- Machine instructions are actually processor-specific strings of 1s and 0s.
- They usually correspond to actual pins and wires on the processor. As such, they are very obscure, very complex and difficult for people to understand.
- Assembler language is simply a programming language that represents various instructions in symbolic code, which is more understandable.

Assembly language programming

- A low-level programming language
- Has a very strong (generally one-to-one) correspondence between the language and the architecture's machine code instructions
- Assembly language is converted into executable machine code by a utility program referred to as an *assembler*

Assembly – pros and cons

Advantages:

- Easier to understand
- Easier to program with/actually possible
- Can use the same programs across a processor family

Disadvantages:

- Hard to cover all the possible machine instructions
- Therefore not optimal
- Very hard to write programs in assembler that will work on different processors.

Murdoch
UNIVERSITY

Summary

Mu
UNIV

Summary

- CPU Organisation
- Instruction Execution
- Design Principles of Modern Computers
- Processor Parallelism
- The Microarchitecture Level
- The Instruction Set Architecture