COMPUTER ORGANIZATION & ARCHITECTURE

RCS-302

Faculties :

1. Mr. Ravi Tiwari (Subject Coordinator)
2. Mr. Gaurav Rajput
3. Mr. Paras Bassi
4. Mr. Vimal Singh

COMPUTER ORGANIZATION & ARCHITECTURE

UNIT 1

Introduction to Computer Organization

Department of Computer Science & Engineering

VISION

To develop competent IT professionals catering to the needs of Industry and society in a global perspective.

MISSION

To attain academic & professional excellence with collective efforts of all stake holders through:

M1: Dissemination of basic concepts and analytical skills.

M2: Exposure to new tools in the area of Information technology.

M3: Effective interaction with industry for better employability.

M4: Inculcating values and professional ethics with social responsibility.

Presentation Outline

- Arithmetic
- Signed and unsigned numbers
- Addition and Subtraction
- Logical operations
- ALU: arithmetic and logic unit
- Multiply
- Divide
- Floating Point
  - notation
  - add
  - multiply

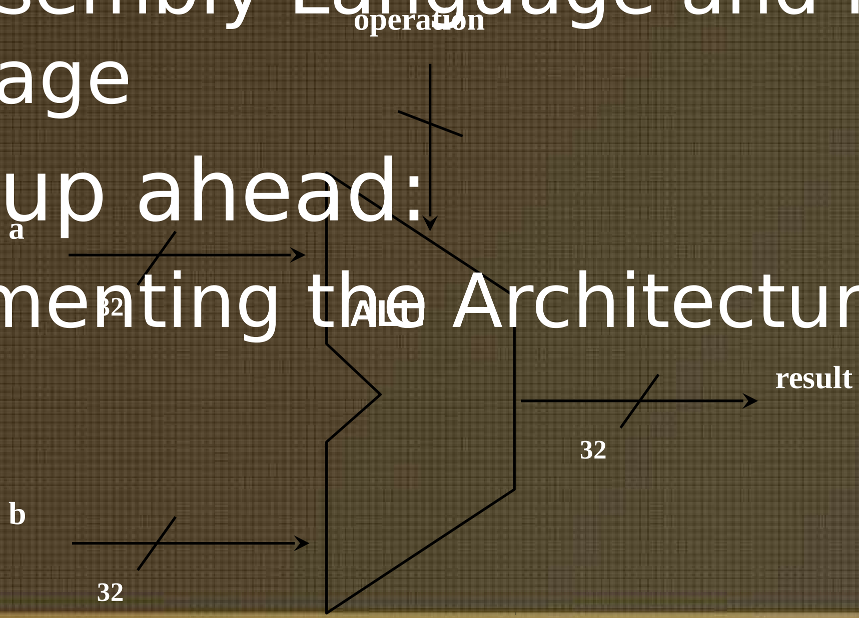
Learning Objectives

The objectives of the following slide is to make student aware about the :

- Arithmetic
- Signed and unsigned numbers
- Addition and Subtraction
- Logical operations
- ALU: Arithmetic and Logic Unit
- Multiply
- Divide
- Floating Point

Arithmetic

- Where we've been:
  - Performance (seconds, cycles, instructions)
  - Abstractions:
    Instruction Set Architecture
    Assembly Language and Machine Language
- What's up ahead:
  - Implementing the Architecture

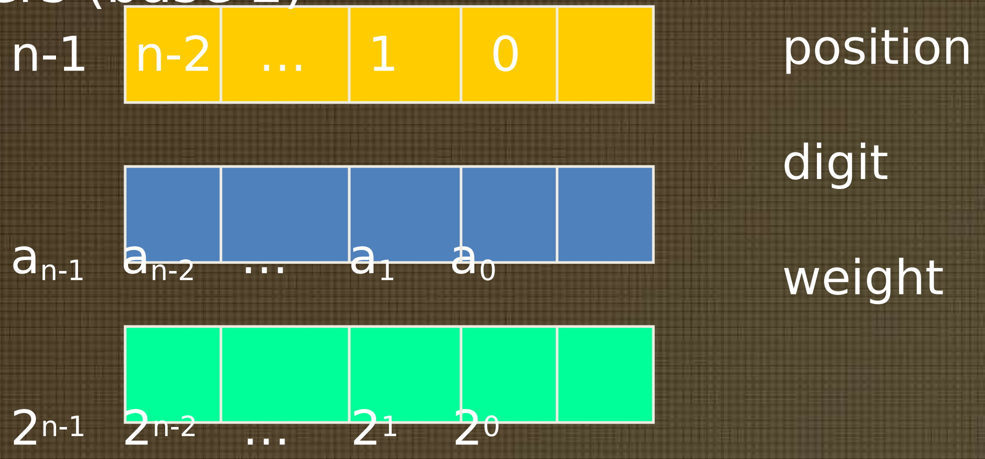


Binary numbers (1)

- Bits have no inherent meaning (no semantics)
- Decimal number system, e.g.:

  $4382 = 4x10^3 + 3x10^2 + 8x10^1 + 2x10^0$
- Can use arbitrary base g; value of digit c at position i:

  $c \text{ x } g^i$
- Binary numbers (base 2)



n-1 n-2 … 1 0 position

a_{n-1} a_{n-2} … a_1 a_0 digit

2^{n-1} 2^{n-2} … 2^1 2^0 weight

- $(a_{n-1}\, a_{n-2} \ldots\, a_1\, a_0)_{\text{two}} = a_{n-1} \text{ x } 2^{n-1} + a_{n-2} \text{ x } 2^{n-2} + \ldots + a_0 \text{ x } 2^0$

Binary numbers (2)

- So far numbers are *unsigned*
- With n bits 2^n possible combinations

1 bit	2 bits	3 bits	4 bits	decimal value
0	00	000	0000	0
1	01	001	0001	1
	10	010	0010	2
	11	011	0011	3
		100	0100	4
		101	0101	5
		110	0110	6
		111	0111	7
			1000	8
			1001	9

- a_0 : *least significant* bit (lsb)
- a_{n-1}: *most significant* bit (msb)

Binary numbers (3)

- Binary numbers (base 2)

  0000 0001 0010 0011 0100 0101 0110 0111 1000 1001...

  decimal: 0...2^n-1

- Of course it gets more complicated:

  - numbers are finite (overflow)
  - fractions and real numbers
  - negative numbers

  - e.g., no MIPS subi instruction;
  - however, addi can add a negative number

How do we represent negative numbers? i.e., which bit patterns will represent which numbers?

Conversion

- Decimaal -> binair

Divide by 2	Remainder
4382	
2191	0
1095	1
547	1
273	1
136	1
68	0
34	0
17	0
8	1
4	0
2	0
1	0
0	1

4382_{ten} =
1 0001 0001 1110_{two}

- Hexadecimal: base 16. Octal: base 8

`1010 1011 0011 1111`$_{two}$ `= ab3f`$_{hex}$

Signed binary numbers

Possible representations:

- Sign Magnitude: One's Complement Two's Complement

000 = +0 000 = +0 000 = +0
001 = +1 001 = +1 001 = +1
010 = +2 010 = +2 010 = +2
011 = +3 011 = +3 011 = +3
100 = -0100 = -3100 = -4
101 = -1101 = -2101 = -3
110 = -2110 = -1110 = -2
111 = -3111 = -0111 = -1

- Issues: balance, number of zeros, ease of operations
- Which one is best? Why?

Two’s complement

- 32 bit signed numbers:

maxint

minint

0000 0000 0000 0000 0000 0000 0000 0000$_{two}$ = 0$_{ten}$
0000 0000 0000 0000 0000 0000 0000 0001$_{two}$ = + 1$_{ten}$
0000 0000 0000 0000 0000 0000 0000 0010$_{two}$ = + 2$_{ten}$
...
0111 1111 1111 1111 1111 1111 1111 1110$_{two}$ = + 2,147,483,646$_{ten}$
0111 1111 1111 1111 1111 1111 1111 1111$_{two}$ = + 2,147,483,647$_{ten}$
1000 0000 0000 0000 0000 0000 0000 0000$_{two}$ = – 2,147,483,648$_{ten}$
1000 0000 0000 0000 0000 0000 0000 0001$_{two}$ = – 2,147,483,647$_{ten}$
1000 0000 0000 0000 0000 0000 0000 0010$_{two}$ = – 2,147,483,646$_{ten}$
...
1111 1111 1111 1111 1111 1111 1111 1101$_{two}$ = – 3$_{ten}$
1111 1111 1111 1111 1111 1111 1111 1110$_{two}$ = – 2$_{ten}$
1111 1111 1111 1111 1111 1111 1111 1111$_{two}$ = – 1$_{ten}$

  – Range $[-2^{31} .. 2^{31} -1]$

- $(a_{n-1}\, a_{n-2} \ldots a_1\, a_0)_{2\text{'s-compl}} = -a_{n-1} \text{ x } 2^{n-1} + a_{n-2} \text{ x } 2^{n-2} + \ldots + a_0 \text{ x } 2^0 = -2^n + a_{n-1} \text{ x } 2^{n-1} + \ldots + a_0 \text{ x } 2^0$

Two's Complement Operations

- Negating a two's complement number: invert all bits and add 1
  - remember: “negate” and “invert” are quite different!

- Proof:

$$a + \bar{a} = 1111.1111b = -1\ d \quad \Rightarrow$$

$$-a = \bar{a} + 1$$

Two's Complement Operations

Converting n bit numbers into numbers with more than n bits:

- – MIPS 8 bit, 16 bit values / immediates converted to 32 bits

- Copy the most significant bit (the sign bit) into the other bits

```
0010   -> 0000 0010
1010   -> 1111 1010
```

- MIPS "sign extension" example instructions:

lb	load byte (signed)
lbu	load byte (unsigned)
slti	set less than immediate (signed)
sltiu	set less than immediate (unsigned)

Addition & Subtraction

- Just like in grade school (carry/borrow 1s)

  0111   0111   0110
  + 0110   - 0110 - 0101

- Two's complement operations easy
  - subtraction using addition of negative numbers

    0110   0110
    - 0101 + 1010

- Overflow (result too large for finite computer word):
  - e.g., adding two n-bit numbers does not yield an n-bit number

    0111
    + 0001   *note that overflow term is somewhat misleading,*
    1000 *it does not mean a carry "overflowed"*

Detecting Overflow

- No overflow when adding a positive and a negative number
- No overflow when signs are the same for subtraction
- Overflow occurs when the value affects the sign:
  - overflow when adding two positives yields a negative
  - or, adding two negatives gives a positive
  - or, subtract a negative from a positive and get a negative
  - or, subtract a positive from a negative and get a positive
- Consider the operations A + B, and A – B
  - Can overflow occur if B is 0 ?
  - Can overflow occur if A is 0 ?

Effects of Overflow

- When an exception (interrupt) occurs:
  - Control jumps to predefined address for exception (*interrupt vector*)
  - Interrupted address is saved for possible resumption in *exception program counter* (EPC); new instruction: `mfc0` `(move from coprocessor0)`
  - *Interrupt handler* handles exception (part of OS). registers `$k0` and `$k1` reserved for OS
- Details based on software system / language
  - C ignores integer overflow; FORTRAN not
- Don't always want to detect overflow

  — new MIPS instructions: `addu`, `addiu`, `subu`

  note: `addiu` and `sltiu` *still sign-extends!*

Logic operations

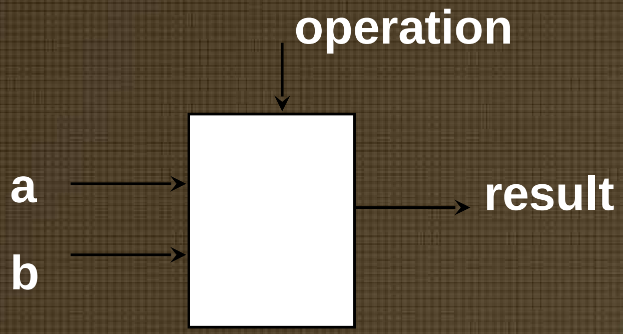
- Sometimes operations on individual bits needed:

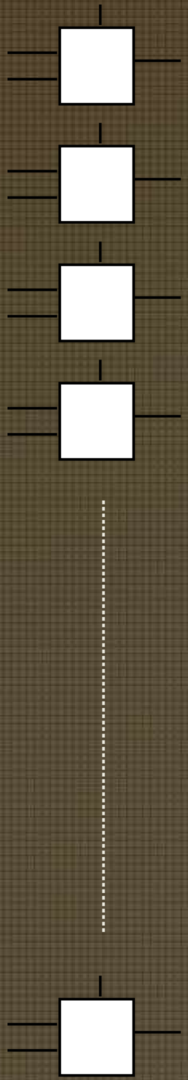
Logic operation	C operation	MIPS instruction
Shift left logical	<<	`sll`
Shift right logical	>>	`srl`
Bit-by-bit AND	`&`	`and, andi`
Bit-by-bit OR	|	`or, ori`

- `and` and `andi` can be used to turn off some bits; `or` and `ori` turn on certain bits
- Of course, AND en OR can be used for logic operations.
  - Note: Language C's logical AND (&&) and OR (||) are *conditional*
- `andi` and `ori` perform no sign extension !

An ALU (arithmetic logic unit)

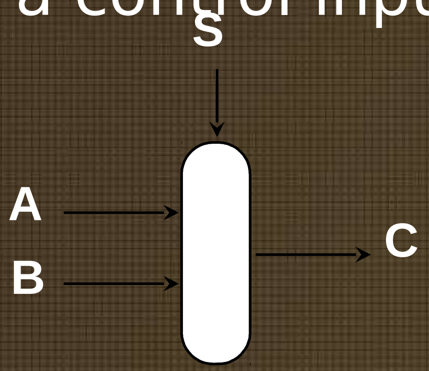
- Let's build an ALU to support the `andi` and `ori` instructions
  - we'll just build a 1 bit ALU, and use 32 of them





Review: The Multiplexor

- Selects one of the inputs to be the output, based on a control input

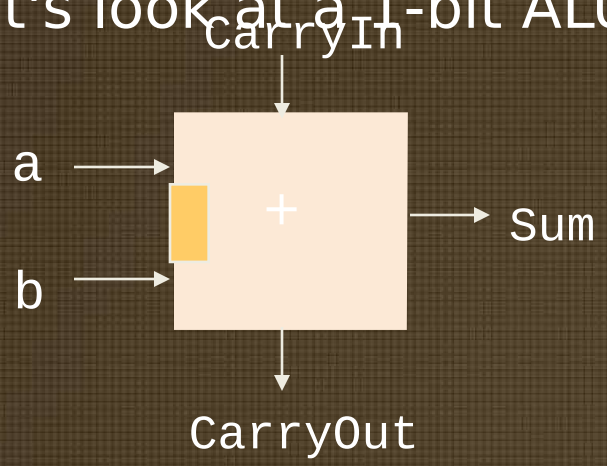


note: we call this a 2-input mux even though it has 3 inputs!

- Lets build our ALU and use a MUX to select the outcome for the chosen operation

Different Implementations

- Not easy to decide the “best” way to build something
  - Don't want too many inputs to a single gate
  - Don’t want to have to go through too many gates
  - For our purposes, ease of comprehension is important
- Let's look at a 1-bit ALU for addition (= full-adder):

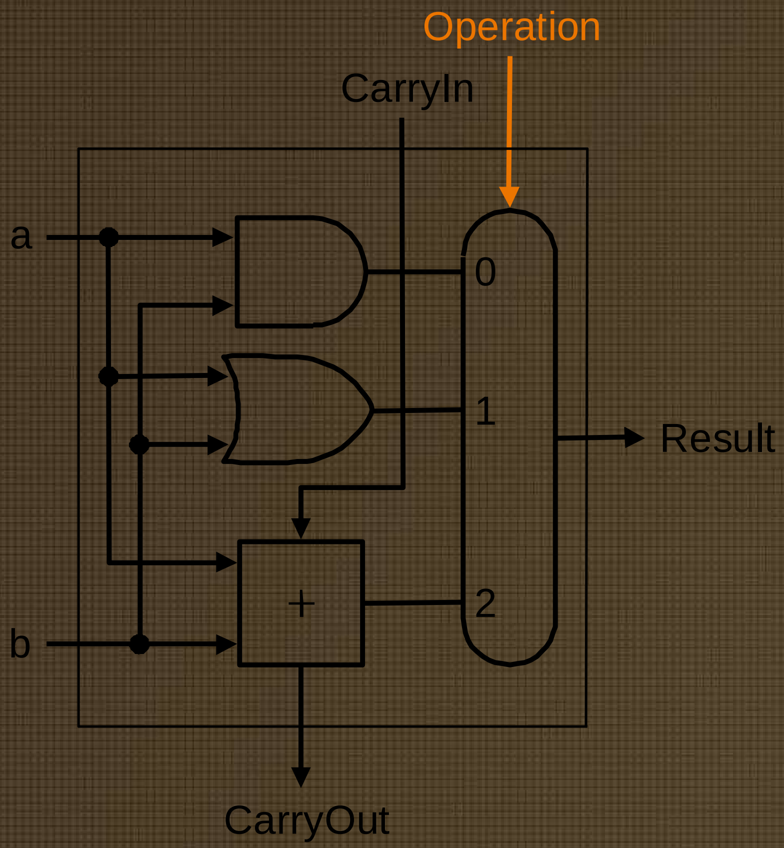


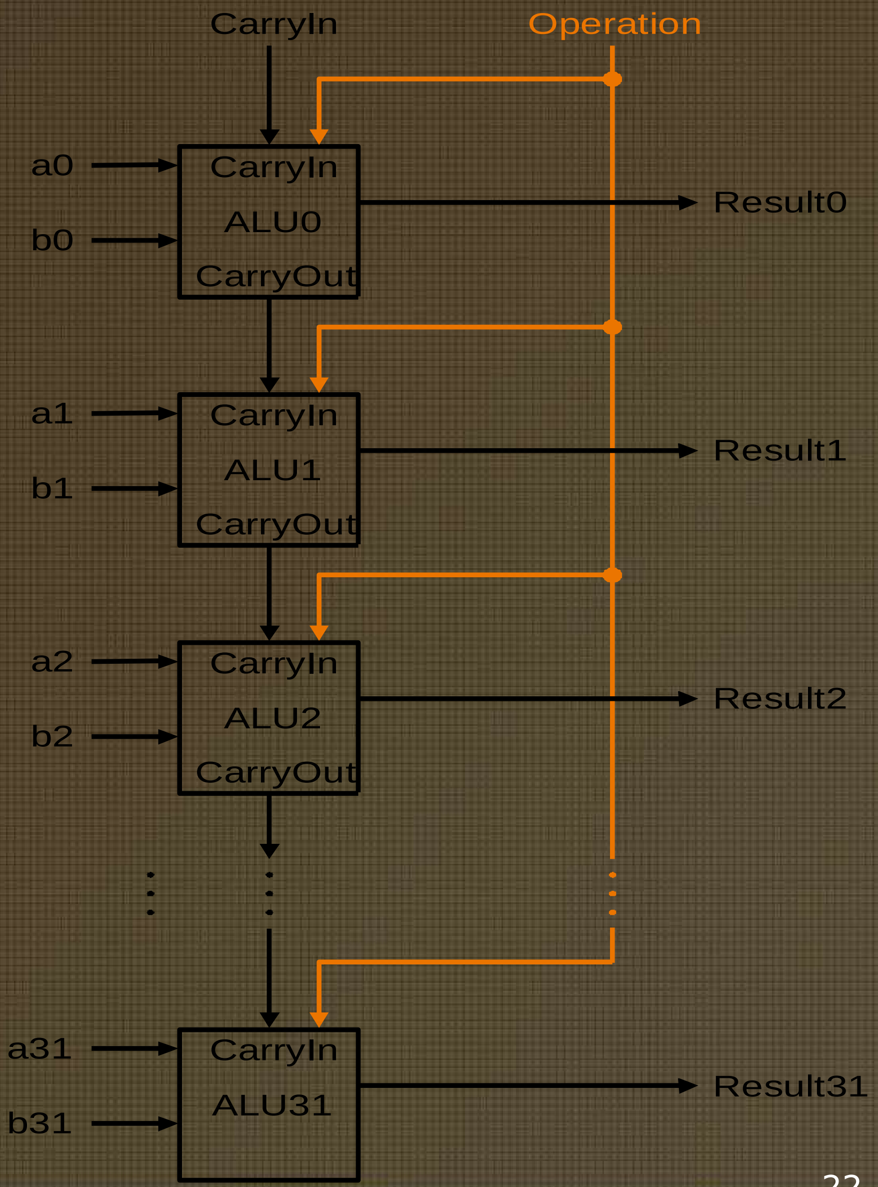
$c_{out} = a\ b + a\ c_{in} + b\ c_{in}$

$sum = a \text{ xor } b \text{ xor } c_{in}$

- How could we build a 1-bit ALU for add, and, and or?
- How could we build a 32-bit ALU?

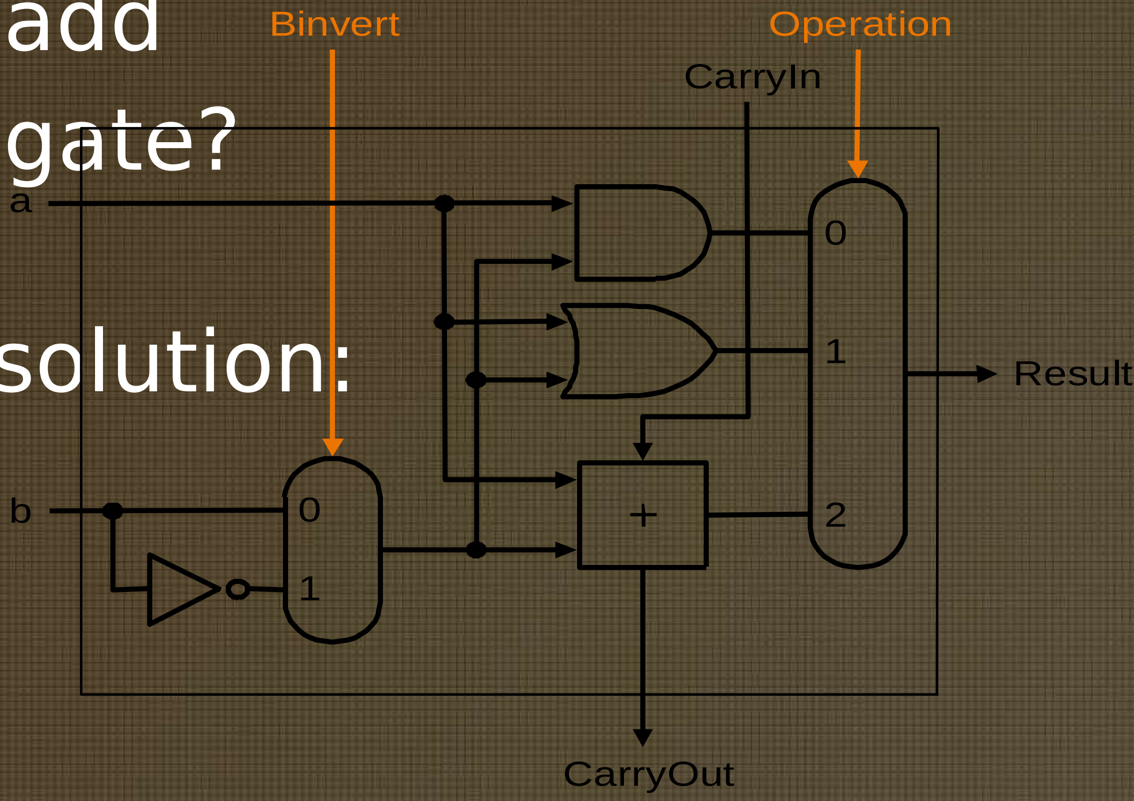
Building a 32 bit ALU



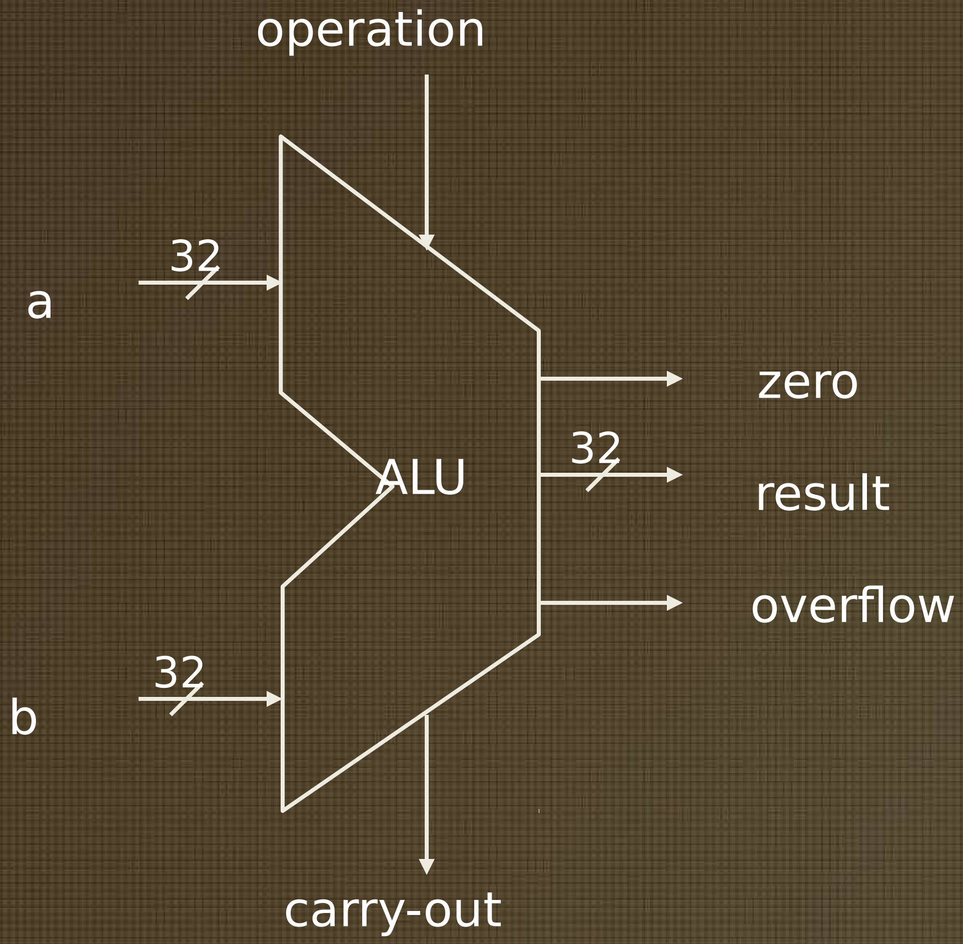


What about subtraction (a – b) ?

- Two's complement approach: just negate **b** and add
- How do we negate?
- A very clever solution:



ALU symbol



Conclusions

- We can build an ALU to support the MIPS instruction set
  - key idea: use multiplexor to select the output we want
  - we can efficiently perform subtraction using two’s complement
  - we can replicate a 1-bit ALU to produce a 32-bit ALU
- Important points about hardware
  - all of the gates are always working
    - not efficient from energy perspective !!
  - the speed of a gate is affected by the number of connected outputs it has to drive (so-called Fan-Out)
  - the speed of a circuit is affected by the number of gates in series

    (on the “critical path” or the “deepest level of logic”)
    - Unit of measure: FO4 = inverter with Fan-Out of 4
    - P4 (heavily superpipelined) has about 15 FO4 critical path

Problem: Ripple carry adder is slow

- Is a 32-bit ALU as fast as a 1-bit ALU?
- Is there more than one way to do addition?
  - Two extremes: ripple carry and sum-of-products
  - How many logic layers do we need for these two extremes?

Can you see the ripple? How could you get rid of it?

$c_1 = b_0c_0 + a_0c_0 + a_0b_0$

$c_2 = b_1c_1 + a_1c_1 + a_1b_1$ $\quad c_2 =$ (..subst c_1..)

$c_3 = b_2c_2 + a_2c_2 + a_2b_2$ $\quad c_3 =$

$c_4 = b_3c_3 + a_3c_3 + a_3b_3$ $\quad c_4 =$

Not feasible! Why not?

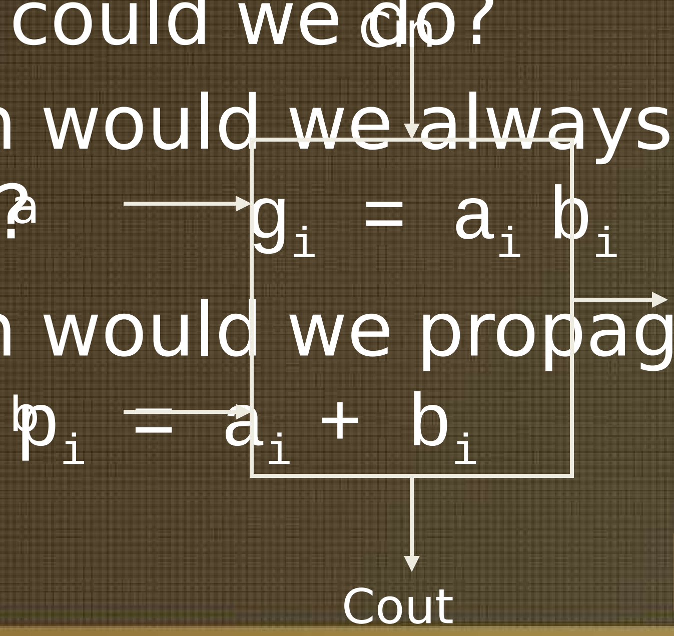
Carry-lookahead adder (1)

- An approach in-between our two extremes
- Motivation:
  - If we didn't know the value of carry-in, what could we do?
  - When would we always generate a carry?

$$g_i = a_i \, b_i$$

  - When would we propagate the carry?

$$p_i = a_i + b_i$$



Cout = Gi + Pi Cin

Carry-lookahead adder (2)

- Did we get rid of the ripple?

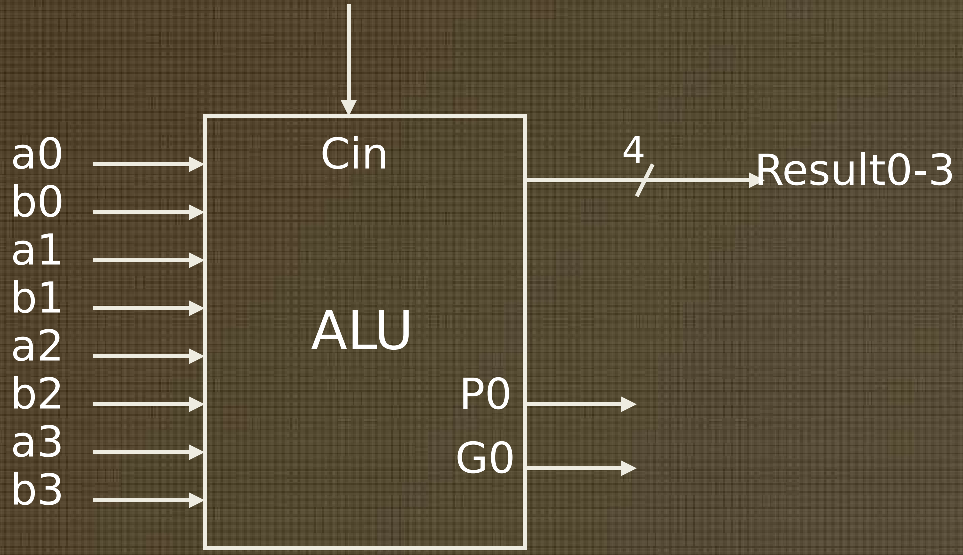
$c_1 = g_0 + p_0c_0$

$c_2 = g_1 + p_1c_1$ $c_2 = g_1 + p_1(g_0 + p_0c_0)$

$c_3 = g_2 + p_2c_2$ $c_3 =$

$c_4 = g_3 + p_3c_3$ $c_4 =$

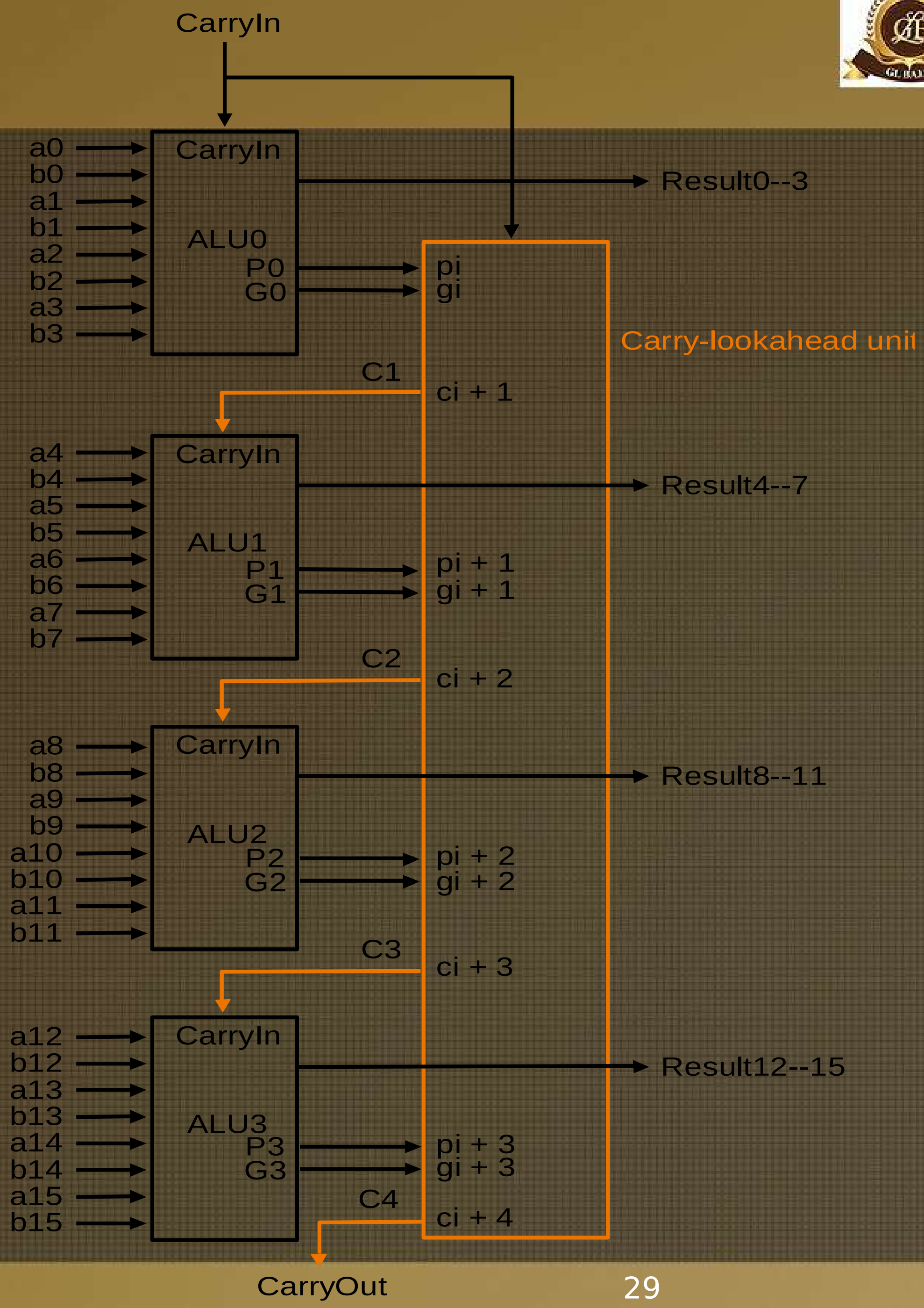
- Feasible ?



$0 = p_0.p_1.p_2.p_3$

$G0= g_3+(p_3.g_2)+(p_3.p_2.g_1)+(p_3.p_2.p_1.g_0)$

- Use principle to build bigger adders
- Can’t build a 16 bit adder this way... (too big)
- Could use ripple carry of 4-bit CLA adders
- Better: use the CLA principle again!



Multiplication (1)

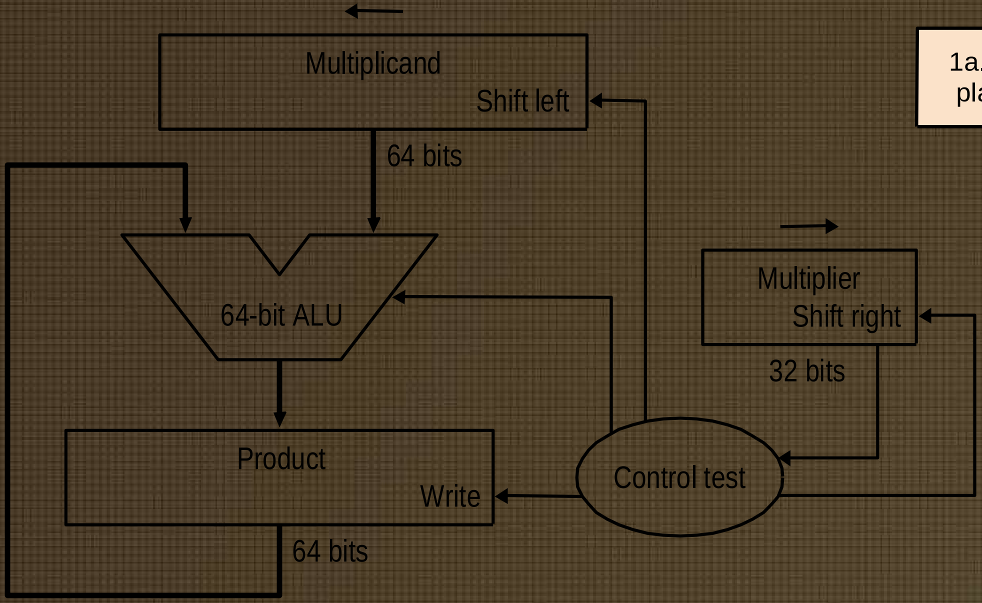
- More complicated than addition
  - accomplished via shifting and addition
- More time and more area
- Let's look at 3 versions based on gradeschool algorithm

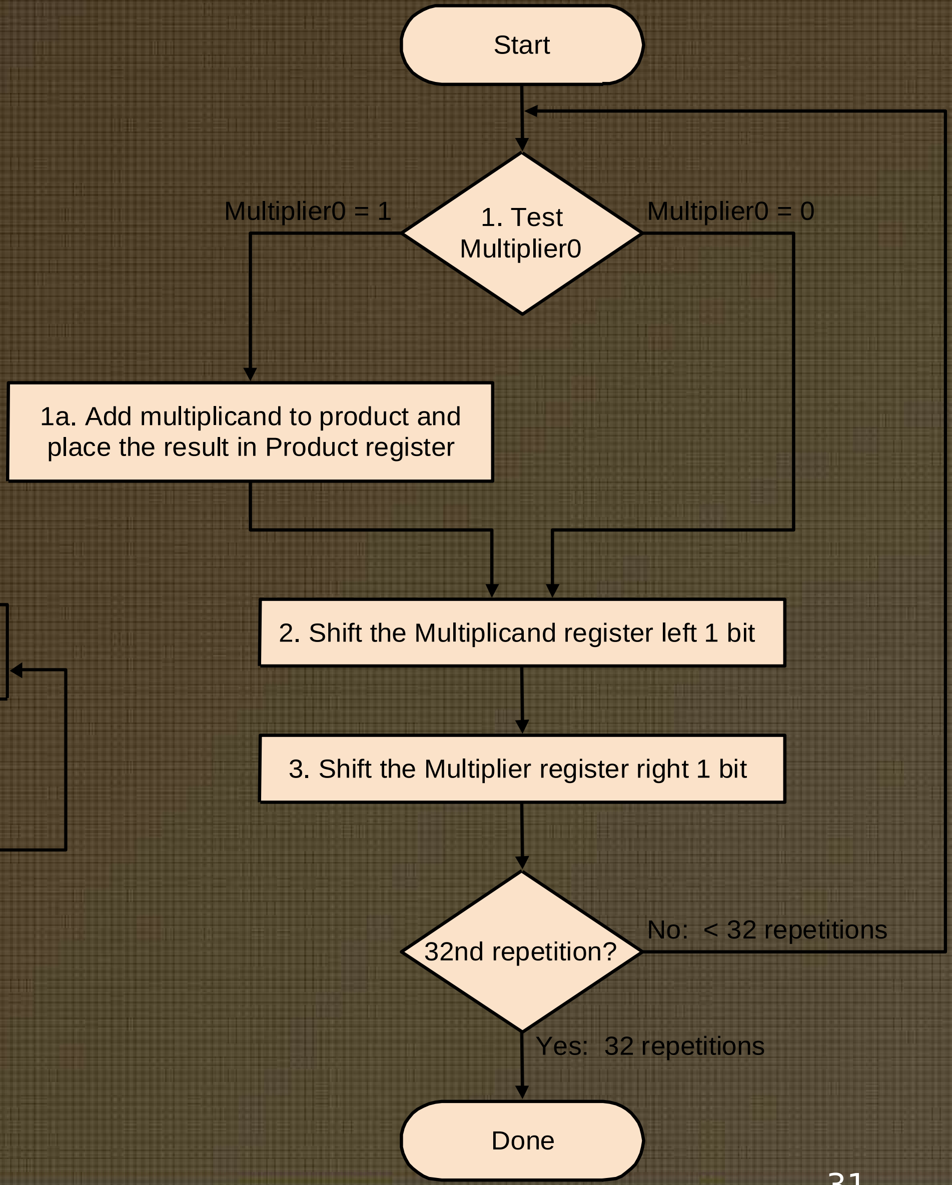
```
      0010   (multiplicand)
___*__1011   (multiplier)
```

- Negative numbers: convert and multiply
  - there are better techniques, we won’t look at them now

Multiplication (2)

First implementation
Product initialized to 0

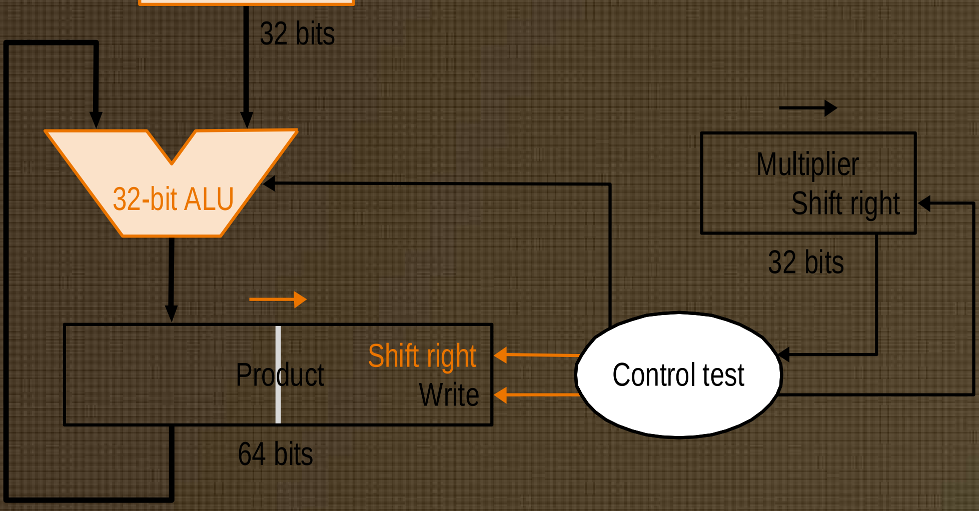


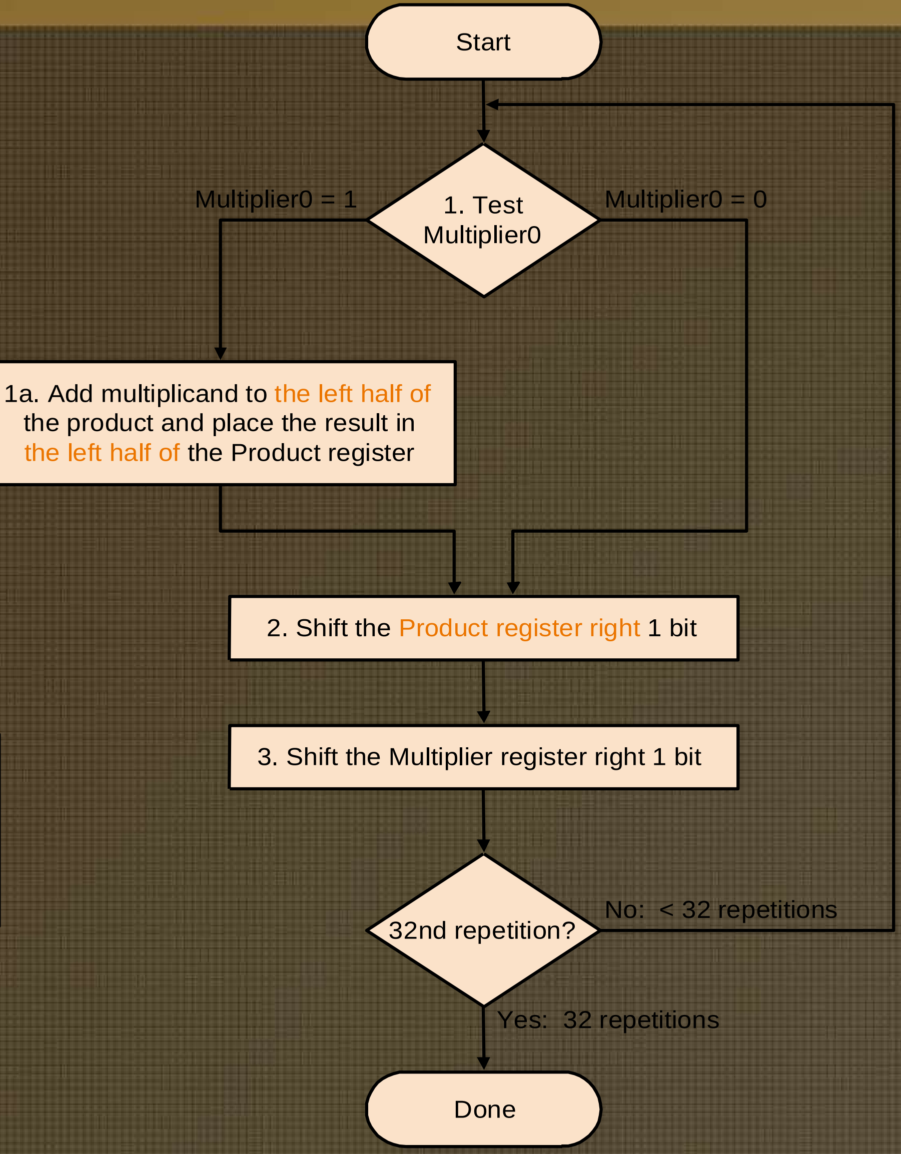


Start
1. Test Multiplier0
Multiplier0 = 1
Multiplier0 = 0
1a. Add multiplicand to product and place the result in Product register
2. Shift the Multiplicand register left 1 bit
3. Shift the Multiplier register right 1 bit
32nd repetition?
No: < 32 repetitions
Yes: 32 repetitions
Done

Multiplication (3)

Second version

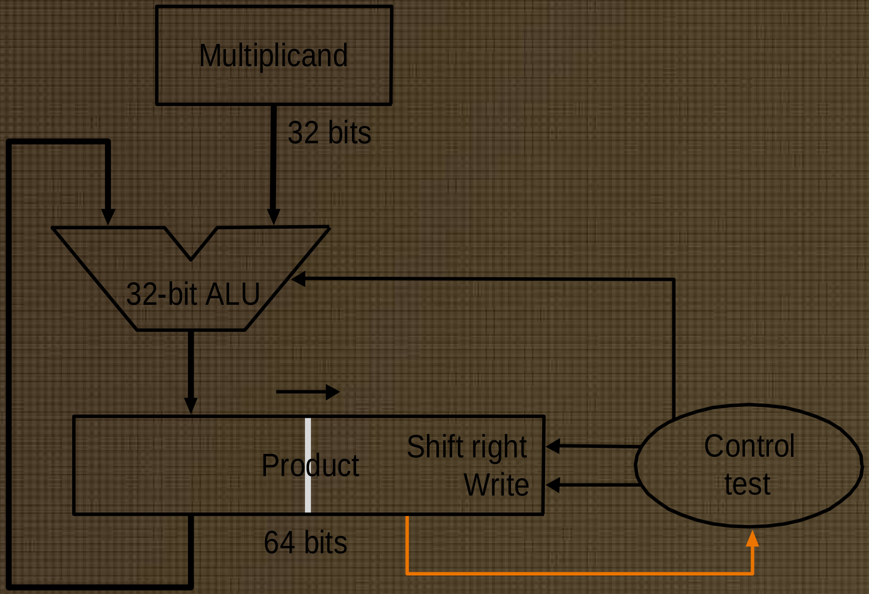


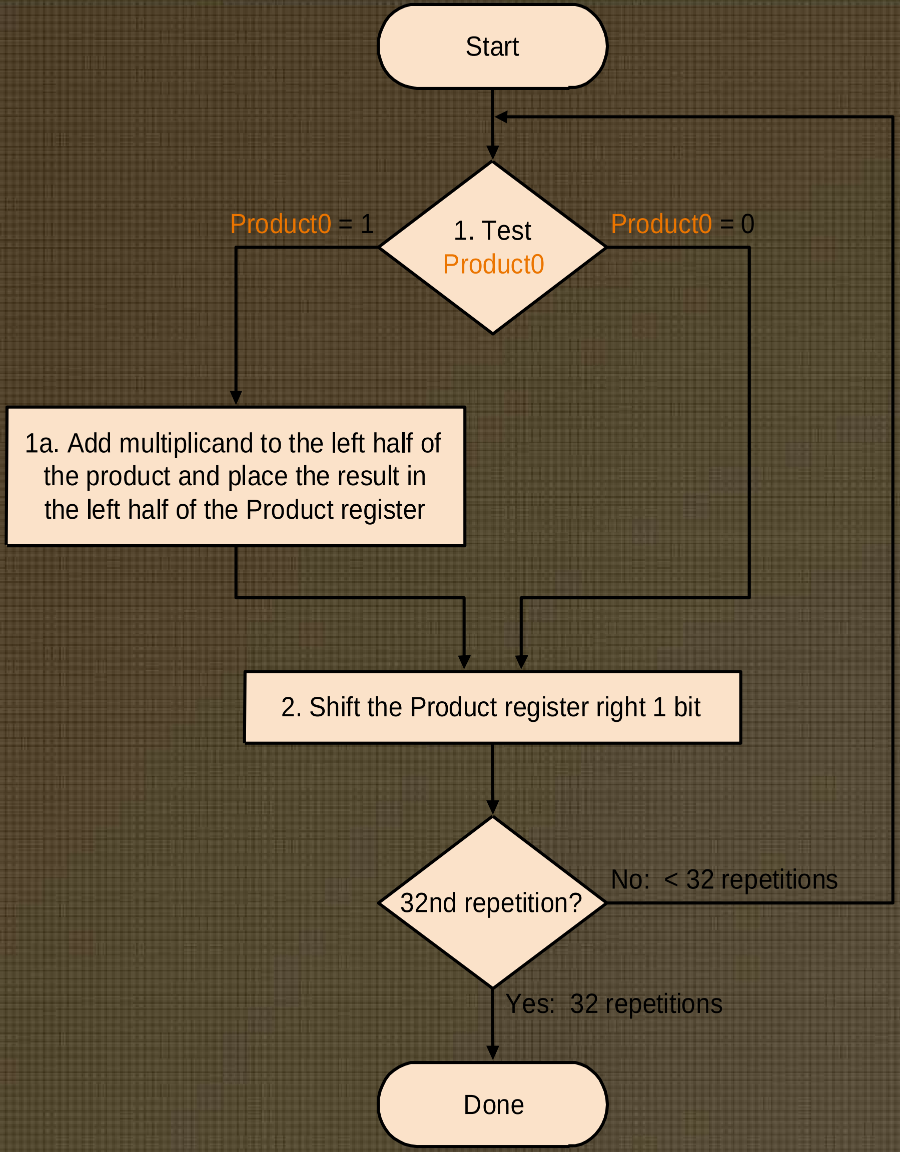


Start
Multiplier0 = 1
1. Test Multiplier0
Multiplier0 = 0
1a. Add multiplicand to the left half of the product and place the result in the left half of the Product register
2. Shift the Product register right 1 bit
3. Shift the Multiplier register right 1 bit
32nd repetition?
No: < 32 repetitions
Yes: 32 repetitions
Done

Multiplication (4)

Final version
Product initialized with multiplier




Start
Product0 = 1
1. Test
Product0
Product0 = 0
1a. Add multiplicand to the left half of the product and place the result in the left half of the Product register
2. Shift the Product register right 1 bit
32nd repetition?
No: < 32 repetitions
Yes: 32 repetitions
Done

Fast multiply: Booth’s Algorithm

- Exploit the fact that: `011111 = 100000 - 1` Therefore we can replace multiplier, e.g.:

  `0001111100  = 0010000000 - 100`
- Rules:

Current bit	*Bit to the right*	*Explanation*	*Operation*
1	0	Begin 1s	Subtract multiplicand
1	1	Middle of 1s	nothing
0	1	End of 1s	Add multiplicand
0	0	Middle of 0s	nothing

Booth’s Algorithm (2)

- Booth’s algorithm works for signed 2’s complement as well (without any modification)
- Proof: let’s multiply b * a
(a$_{i-1}$ - a$_i$) indicates what to do:

0 : do nothing
+1 : add b
-1 : subtract

We get b*a =

$$\sum_{i=0}^{31} (a_{i-1} - a_i) * b * 2^i =$$

$$b * \left[a_{31} * -2^{31} + \sum_{i=0}^{30} a_i * 2^i \right]$$

This is exactly what we need !

Division

- Similar to multiplication: repeated subtract
- The book discusses again three versions

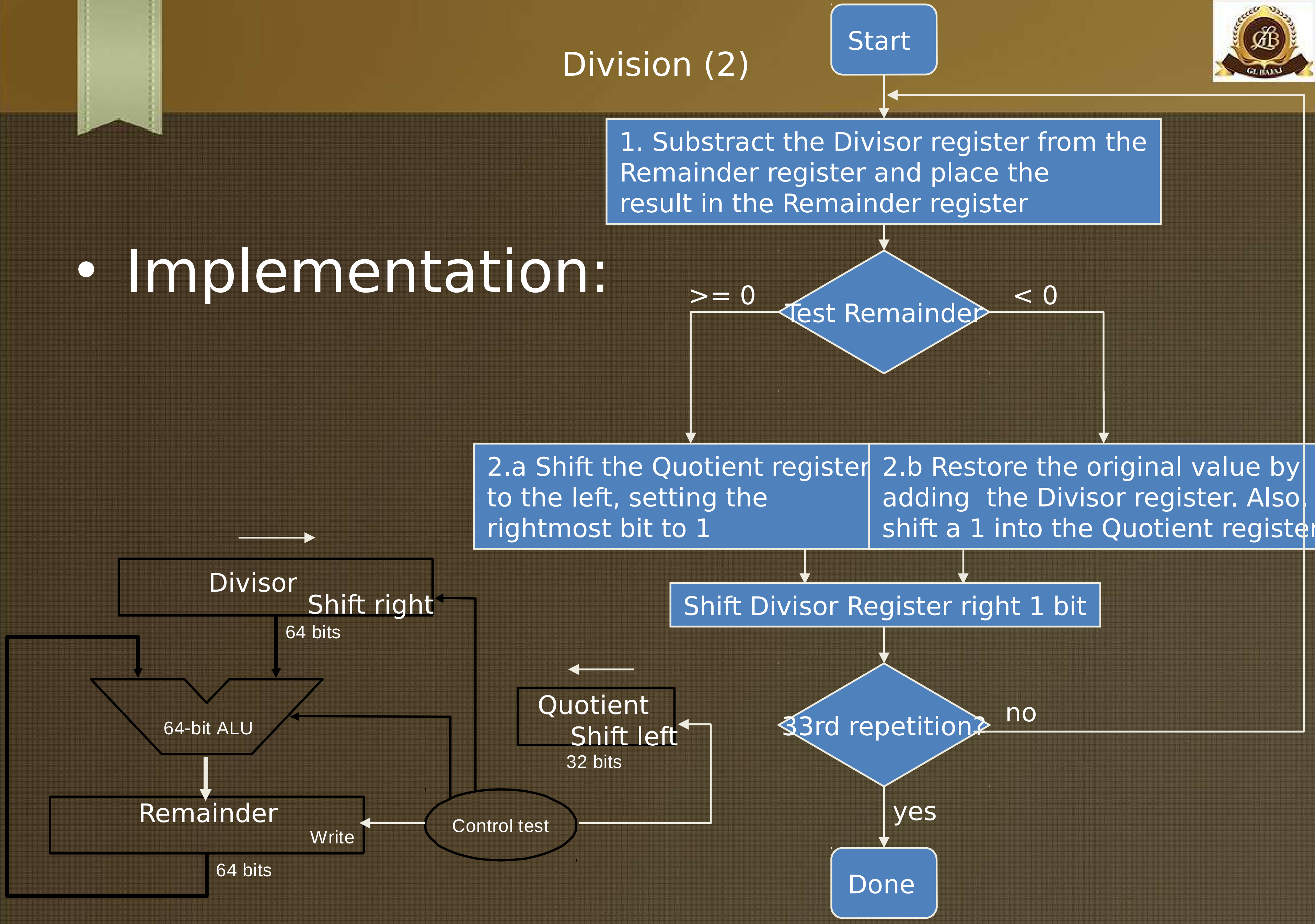
Divide (1)

- Well known algorithm:

```
                    Dividend
   Divisor  1000/1001010\1001  Quotient
                   -1000
                         10
               101
               1010
              -1000
                  10   Remainder
```

Division (2)

- Implementation:



Start

1. Substract the Divisor register from the Remainder register and place the result in the Remainder register

Test Remainder

2.a Shift the Quotient register to the left, setting the rightmost bit to 1

2.b Restore the original value by adding the Divisor register. Also, shift a 1 into the Quotient register

Shift Divisor Register right 1 bit

33rd repetition?

Done

Multiply / Divide in MIPS

- MIPS provides a separate *pair of 32-bit registers* for the result of a multiply and divide: Hi and Lo

```
mult  $s2,$s3    # Hi,Lo = $s2 * $s3
div   $s2,$s3    # Hi,Lo = $s2 mod $s3,
$s2 / $s3
```

- Copy result to general purpose register

```
mfhi  $s1           # $s1 = Hi
mflo  $s1           # $s1 = Lo
```

- There are also unsigned variants of mult and div: `multu` and `divu`

Shift instructions

- `sll`
- `srl`
- `sra`

- Why not ‘`sla`’ instruction ?

Shift: a quick way to multiply and divide with power of 2 (strength reduction). Is this always allowed?

Floating Point (a brief look)

- We need a way to represent
  - numbers with fractions, e.g., 3.1416
  - very small numbers, e.g., .000000001
  - very large numbers, e.g., 3.15576×10^9
- Representation:
  - sign, exponent, significand: $(–1)^{sign} \times$ significand $\times 2^{exponent}$
  - more bits for significand gives more accuracy
  - more bits for exponent increases range
- IEEE 754 floating point standard:
  - single precision : 8 bit exponent, 23 bit significand
  - double precision: 11 bit exponent, 52 bit significand

IEEE 754 floating-point standard

- Leading “1” bit of significand is implicit

- Exponent is “biased” to make sorting easier
  - all 0s is smallest exponent all 1s is largest
  - bias of 127 for single precision and 1023 for double precision
  - summary: $(-1)^{\text{sign}} \times (1+\text{significand}) \times 2^{\text{exponent - bias}}$

- Example:
  - decimal: -.75 = -3/4 = $-3/2^2$
  - binary : -.11 = -1.1×2^{-1}
  - floating point: exponent = -1+bias = 126 = 01111110
  - IEEE single precision:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Floating Point Complexities

- Operations more complicated: align, renormalize, ...
- In addition to overflow we can have “underflow”
- Accuracy can be a big problem
  - IEEE 754 keeps two extra bits, guard and round, and additional sticky bit (indicating if one of the remaining bits unequal zero)
  - four rounding modes
  - positive divided by zero yields “infinity”
  - zero divide by zero yields “not a number”
  - other complexities
- Implementing the standard can be tricky
- Not using the standard can be even worse
  - see text for description of 80x86 and Pentium bug!

Conversion: decimal IEEE 754 FP

- Decimal number (base 10)

  123.456 = $1x10^{2}+2x10^{1}+3x10^{0}+4x10^{-1}+5x10^{-2}+6x10^{-3}$

- Binary number (base 2)

  101.011 = $1x2^{2}+0x2^{1}+1x2^{0}+0x2^{-1}+1x2^{-2}+1x2^{-3}$

- Example conversion: 5.375
  - Multiply with power of 2, to get rid of fraction:

    5.375 = 5.375x16 / 16 = 86 x 2^{-4}
  - Convert to binary, and normalize to 1.xxxxx

    86 x 2^{-4} = 1010110 x 2^{-4} = 1.01011 x 2^{2}
  - Add bias (127 for single precision) to exponent:

    exponent field = 2 + 127 = 129 = 1000 0001
  - IEEE single precision format (remind the leading “1” bit):

0	10000001	01011000000000000000000
sign	exponent	significand

Assignment 1

Q1. What is Computer Architecture?

Q2. Convert the number (-16.125) into IEEE 754 single precision format.

Q3. Discuss about hardware used for signed operand multiplication.

Q4. Explain advantages and disadvantages of daisy chaining and polling bus arbitration scheme.

Q5.Discuss the importance of Array Multiplier. Explain your answer with a 3 bit to 2 bit array multipliers.

Tutorial 1

Q1. Convert the following arithmetic expressions from infix to reverse polish notation

i) A* B + C * D + E * F

ii) AB + A (BD + CE)

Q2. Convert the number (-16.125) into IEEE 754 single precision format.

Q3.Explain the floating-point addition/subtraction algorithm with flow chart.

Q4. What do you mean by Array Multiplier? Explain 3bit by 2 bit array multiplier.

Q5. What do you mean by Booth's Algorithm? Discuss the required hardware and iterate your algorithm for the product (+8)*(-3).

Outcomes

After reading above topics students will be able to:

- An ability to apply the knowledge of mathematics, science, and engineering in the field of Information Technology for the solution of engineering problems.
- Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

COMPUTER ORGANIZATION & ARCHITECTURE

UNIT 2

Instructions Cycle & Control Unit

Presentation Outline

- Instructions
  - Types & Format
- Cycles
- Control Unit
  - Hardwired
  - Microprogram

Learning Objectives

The objectives of the following slide is to make student aware about the :

- Instructions & their types .
- Instruction Cycles
- Control Unit
  - Hardwired
  - Microprogram

INSTRUCTION

Instruction is command which is given by the user to computer.

Instruction cycle

- The time period during which one instruction is fetched from memory and execute when a computer given an instruction in machine language.
- Each instruction is further divided into sequence of phases.
- After the execution the program counter is incremented to point to the next instruction.

Phases

- Fetch an instruction from memory
- Decode the instruction
- Execute the instruction

Fetch cycle

- In this phase the sequence counter is initialized to 0.
- The address of first instruction from PC is loaded into address register during the first clock cycle.

The Fetch Cycle

- Consists of three time units and four *micro-operations*.
- Each *micro-operation* involves the movement of data into or out of a register.

$$t_1 : PC \rightarrow MAR$$

$$t_2 : MEMORY \rightarrow MDR$$

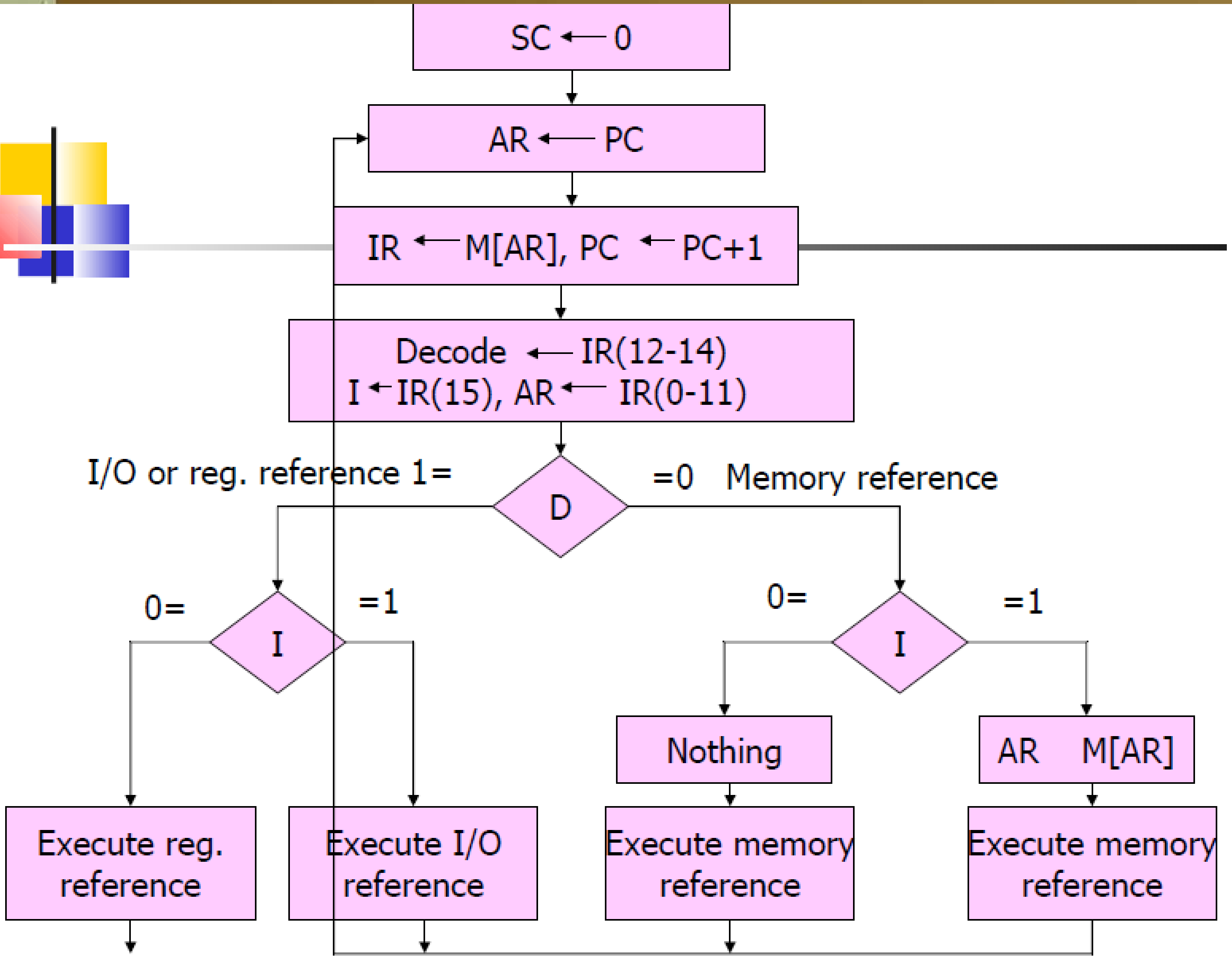
$$t_3 : PC + 1 \rightarrow PC$$

$$MDR \rightarrow IR$$

Decode cycle

- The instruction is decoded by the instruction decoder of a processor.

- All the bits of the instruction under execution stored in IR are analyzed and decode in third clock cycle.



SC ← 0

AR ← PC

IR ← M[AR], PC ← PC+1

Decode ← IR(12-14)
I ← IR(15), AR ← IR(0-11)

I/O or reg. reference 1=

D

=0 Memory reference

0=

I

=1

0=

I

=1

Nothing

AR M[AR]

Execute reg. reference

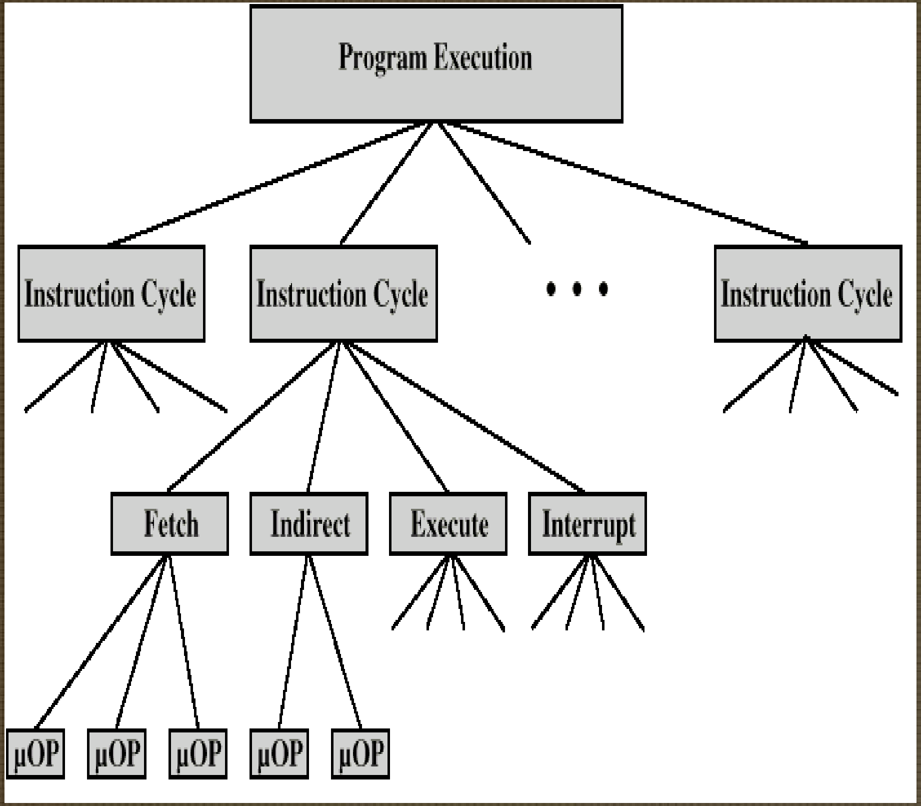
Execute I/O reference

Execute memory reference

Execute memory reference

Micro-operations

- Are the functional, or atomic, operations of a processor.
- A single *micro-operation* generally involves a transfer between registers, transfer between registers and external bus, or a simple ALU operation.



Micro-operations and the Clock

- Each clock pulse defines a time unit, which are of equal duration.
- *Micro-operations* are performed within this time unit.
- If multiple *micro-operations* do not interfere with one another then grouping of micro-operations can be performed within one time unit.
- Grouping can be performed as long as;
  - Proper sequence of events are followed
    - PC MAR must be done first in order for MEMORY MDR
  - Conflicts are avoided
    - MEMORY MDR can not be in the same time unit as MDR IR

The Indirect Cycle

- Occurs if the instruction specifies an indirect address.
- Consists of three time unit and three *micro-operations*.
- Data is transferred to the MAR from the IR, which is used to fetch the address of the operand, the IR is then updated from MDR so it contains a direct address rather than indirect.

$$t_1 : IR \longrightarrow MAR$$

$$t_2 : MEMORY \longrightarrow MDR$$

$$t_3 : MDR \longrightarrow IR$$

Types of Micro-operation

- Transfer data between registers
- Transfer data from register to external
- Transfer data from external to register
- Perform arithmetic or logical operations

Functions of Control Unit

- *Sequencing*

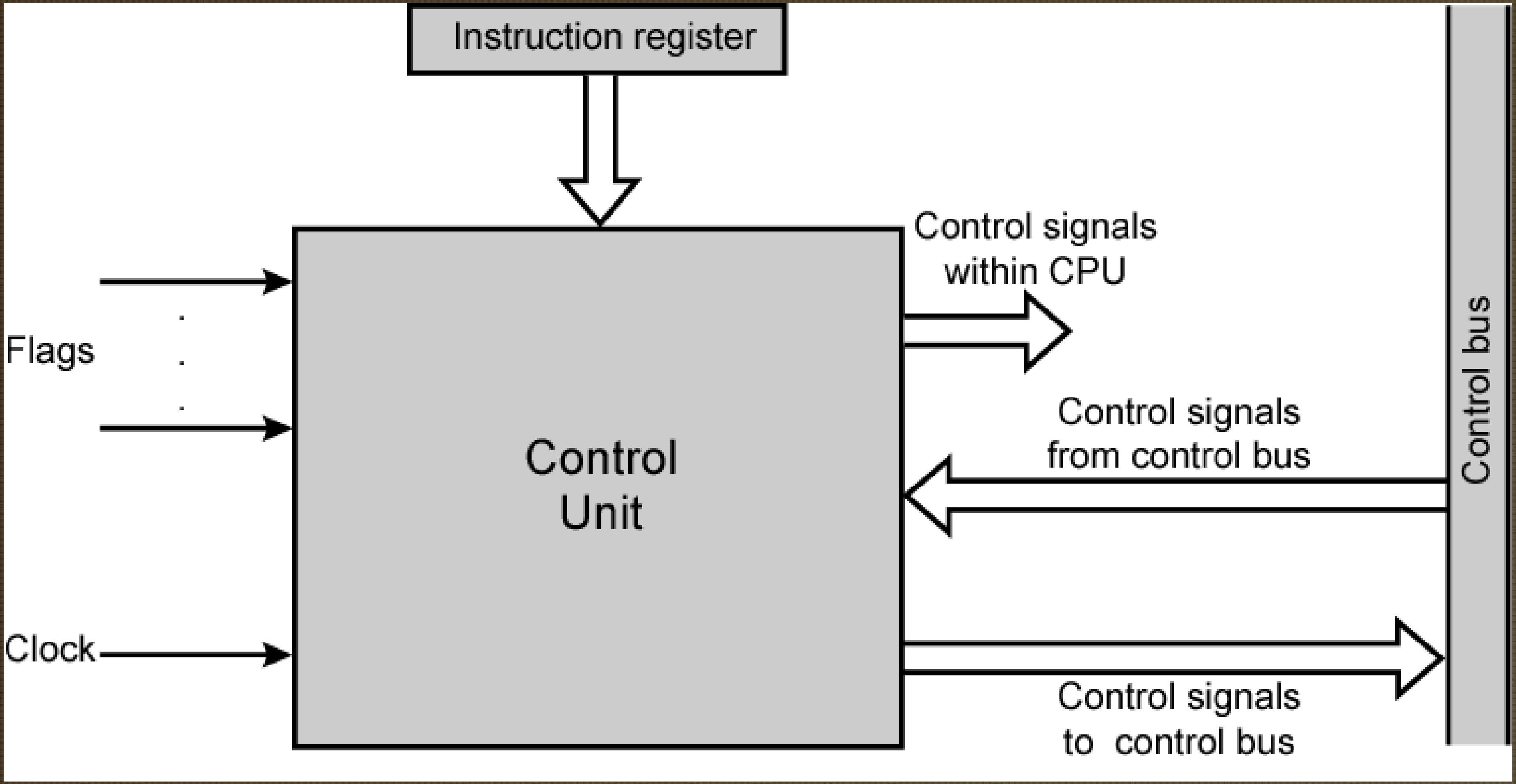
  Causes the processor to step through a series of micro-operations

- *Execution*

  Causes the performance of each micro-operation

- *This is done using Control Signals*

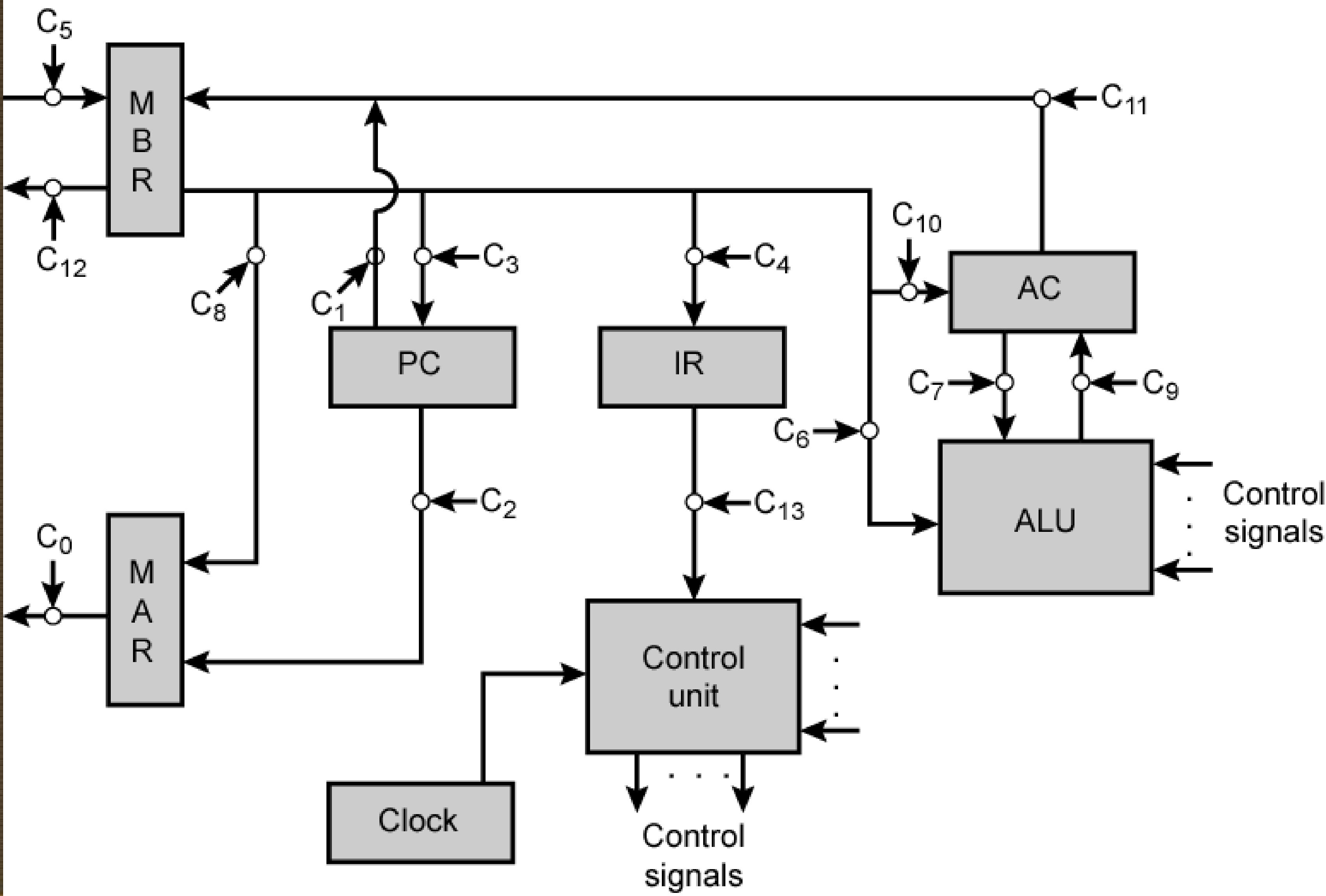
Model of Control Unit



Control Signals - Input

- *Clock*
  - One micro-instruction (or set of simultaneous micro instructions) per clock pulse.
- *Instruction register*
  - Op-code of the current instruction
  - Determines which micro-instructions are performed
- *Flags*
  - Determines the status of the processor
  - Results of previous ALU operations
- *Control Signals from control bus*
  - Interrupts
  - Acknowledgements

Data Paths and Control Signals



PROCESSOR ORGANIZATION

- Organization is how features are implemented
  - Control signals, interfaces, memory technology.
  - e.g. Is there a hardware multiply unit or is it done by repeated addition

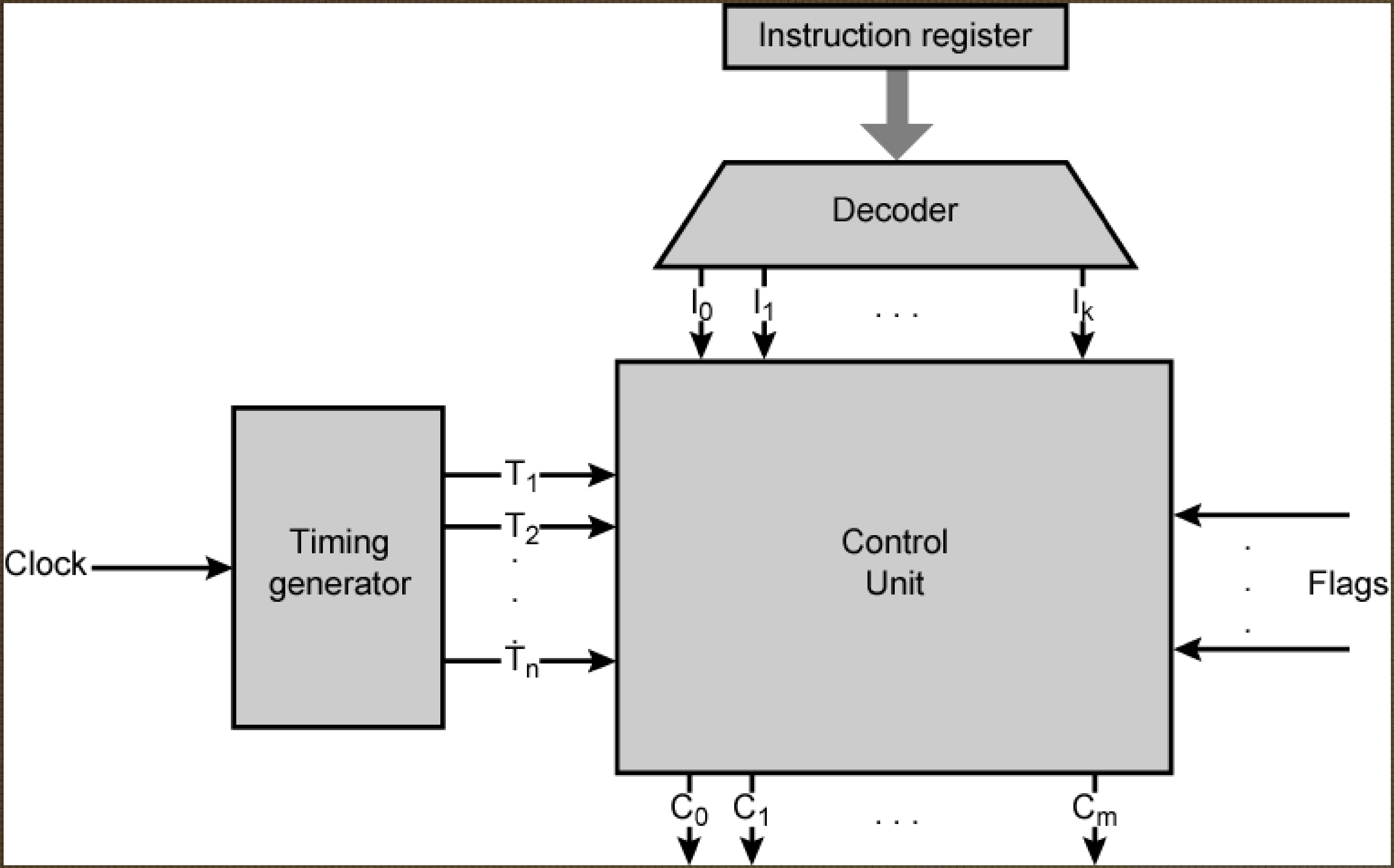
Internal Organization

- Usually a single internal bus
- Using single bus simplifies & saves space
- Gates control movement of data onto and off the bus
- Control signals control data transfer to and from external systems bus
- Temporary registers needed for proper operation of ALU

Hardwired Implementation

- Control unit inputs
- Flags and control bus
  - Each bit means something
- Instruction register
  - Op-code causes different control signals for each different instruction
  - Decoder takes encoded input and produces single output
  - *n* binary inputs and 2^n outputs

Control Unit with Decoded Inputs



Hardwired Logic

- Logic Gates Hardwired Internally
  - Functions predefined
  - Truth Tables
  - Boolean Logic used to define timing
  - Connect Instructions
  - Unique logic for each set of op-codes

Problems With Hard Wired Designs

- Complex sequencing & micro-operation logic
- Difficult to design and test
- Inflexible design
- Difficult to add new instructions

Assignment 2

Q1. List and explain different type of shift micro operation.

Q2. Draw the flow chart for the execution of a complete instruction in a basic computer.

Q3. An instruction format, there are 16 bits in an instruction word. Bit 0 to 11 convey the address of memory location for memory related instructions. For non-memory instructions these bits convey various registers or I/O operations. Bits 12 to 14 show the various memory operations such as AND, ADD, LDA etc. Bit 15 shows if the memory accessed directly or indirectly. For such an instruction format draw the block diagram of control unit of a computer and briefly explain how an instruction decoded and executed by this control unit.

Q4. What is subroutine call ? Explain with an example.

Q5. What is micro instruction. Explain the working of micro program sequencer with diagram.

Tutorial 2

Q1. Write a program to evaluate the arithmetic statement

X=(A-B+C*(D*E-F))/(G+H*K)

i. Using a general register computer with one address instructions.

ii. Using an accumulator type computer with zero address instruction.

Q2. An instruction is stored at location 300 with its adder field at 301 with its adder field at 301. The adder field has the value 400. A processor register R1 contains the number 200. Evaluate the effective address if the addressing mode of the instruction is: i) Direct ii) Immediate iii) Relative

Q3. What is addressing mode? Explain different types with diagram.

Q4. Differentiate horizontal and vertical microprogramming .

Q5. What is control word?

Outcomes

After reading above topics students will be able to:

- Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

COMPUTER ORGANIZATION & ARCHITECTURE

UNIT 3

Memory Organization

Presentation Outline

- Memory
  - Types
  - Concepts
  - Hierarchy
- RAM Organization
- Cache Memory & Performance
- Mapping Techniques
- Virtual Memory

Learning Objectives

The objectives of the following slide is to make student aware about the :

- Memory Types, Concepts, Hierarchy
- RAM Organization
- Cache Memory & Performance
- Mapping Techniques
- Virtual Memory Implementation

Multiple-Chip SRAM

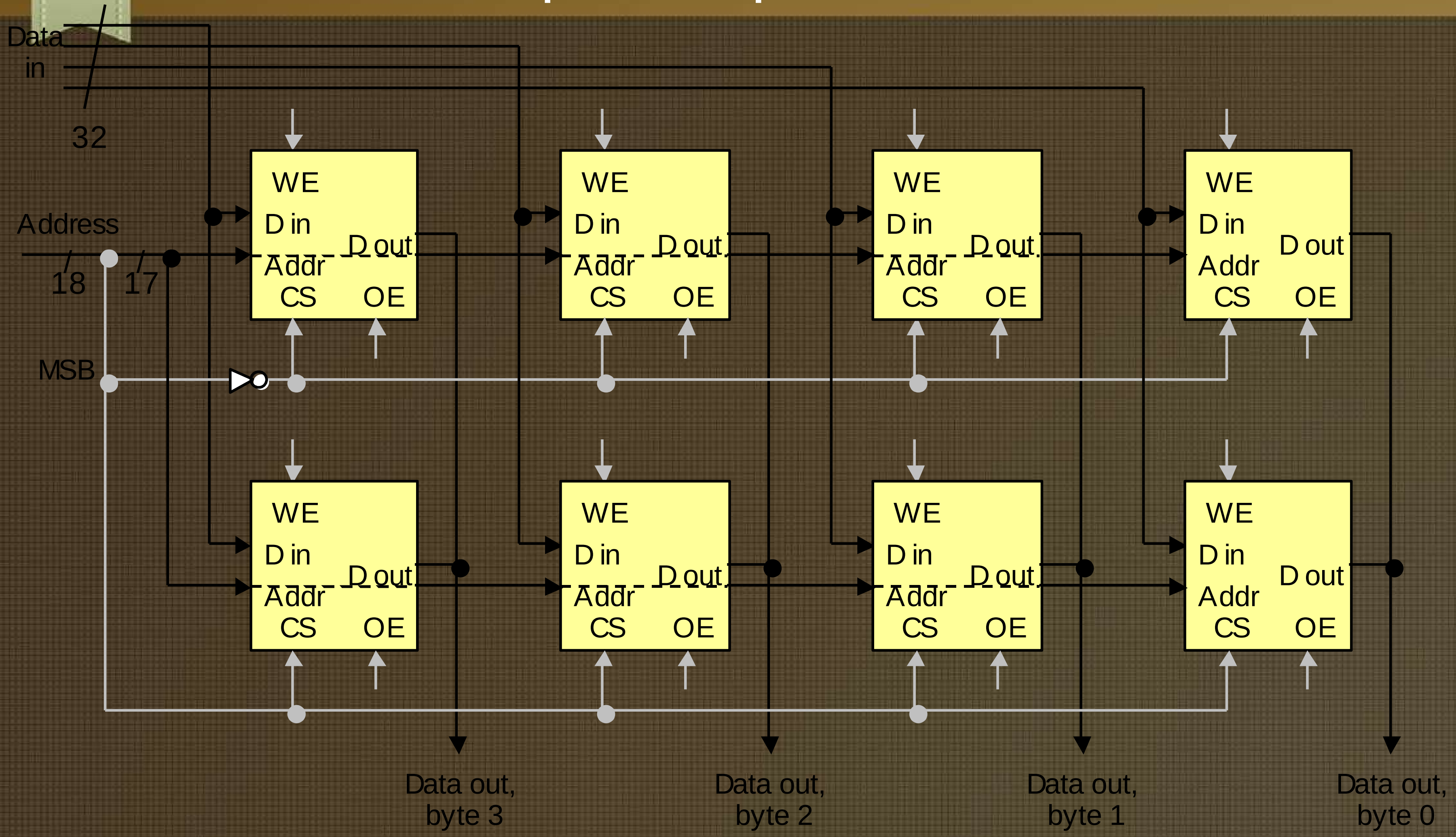


Fig. 17.2 Eight 128K × 8 SRAM chips forming a 256K × 32 memory unit.

SRAM with Bidirectional Data Bus

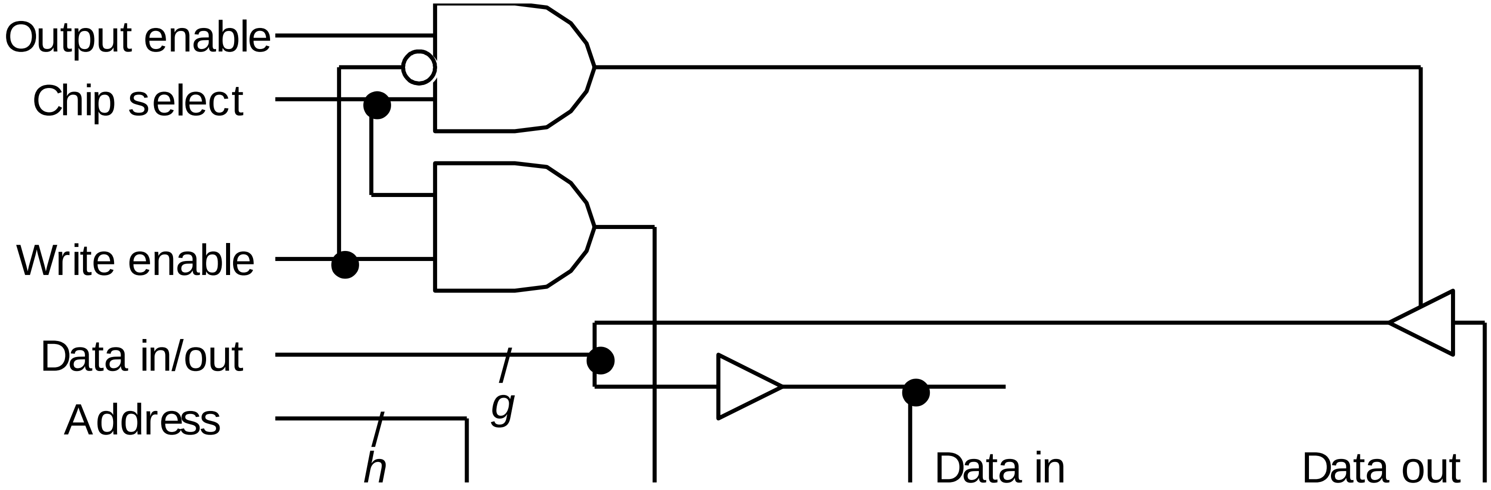


Fig. 17.3 When data input and output of an SRAM chip are shared or connected to a bidirectional data bus, output must be disabled during write operations.

DRAM and Refresh Cycles

DRAM vs. SRAM Memory Cell Complexity

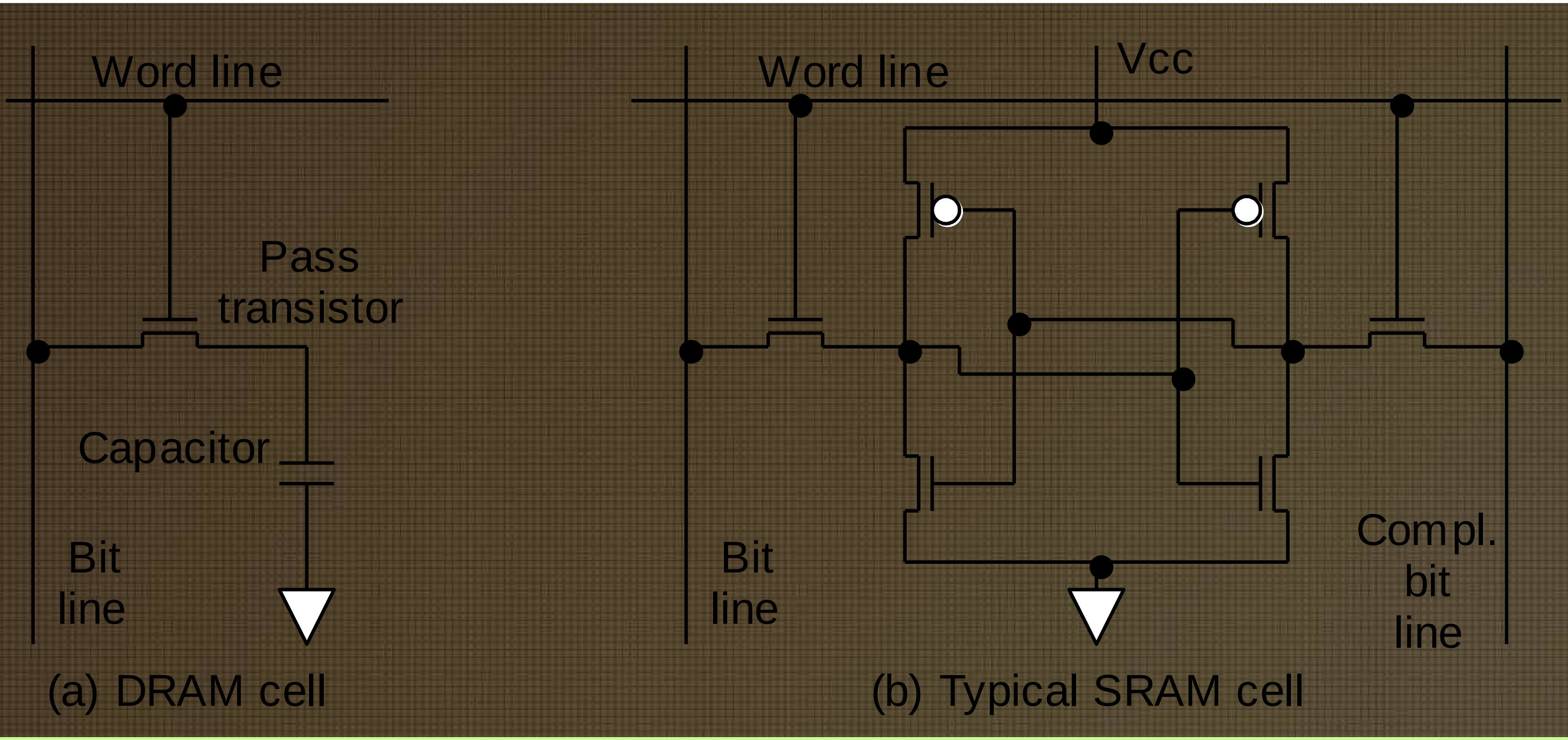


Fig. 17.4 Single-transistor DRAM cell, which is considerably simpler than SRAM cell, leads to dense, high-capacity DRAM memory chips.

Bridging the CPU-Memory Speed Gap

Idea: Retrieve more data from memory with each access

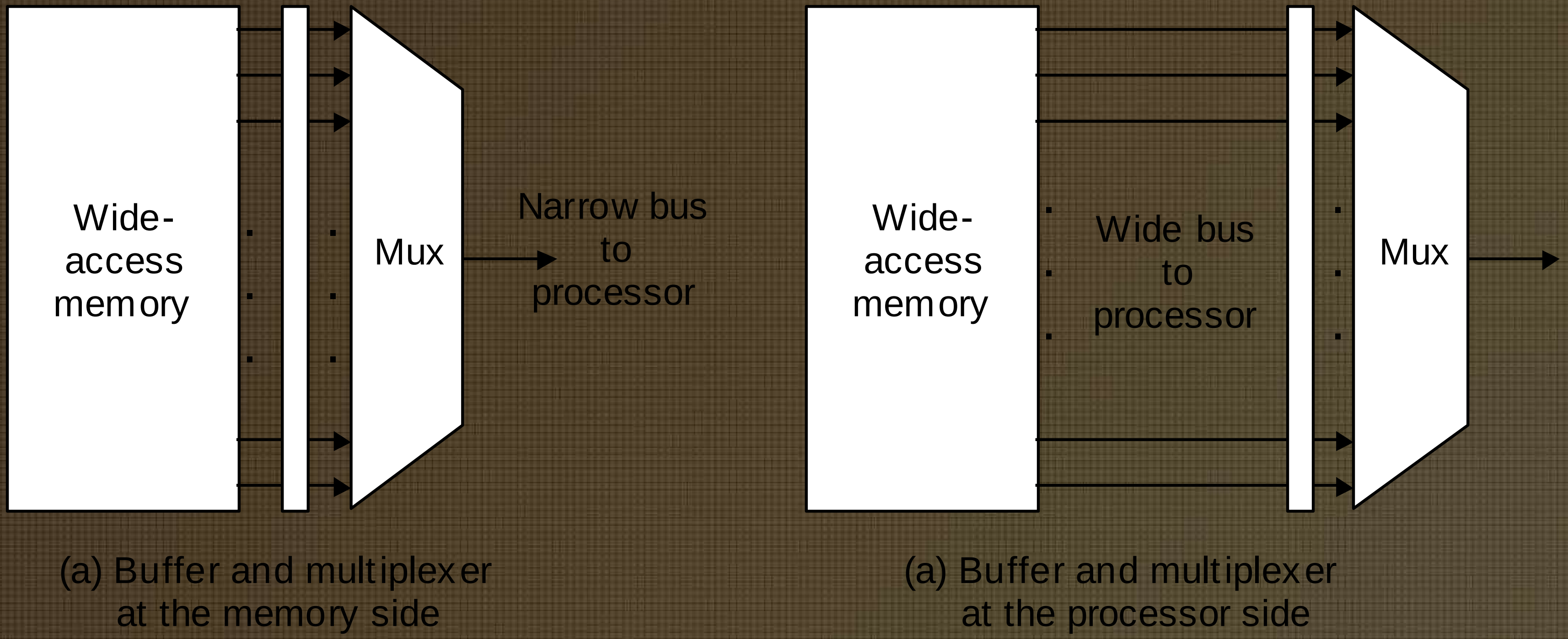
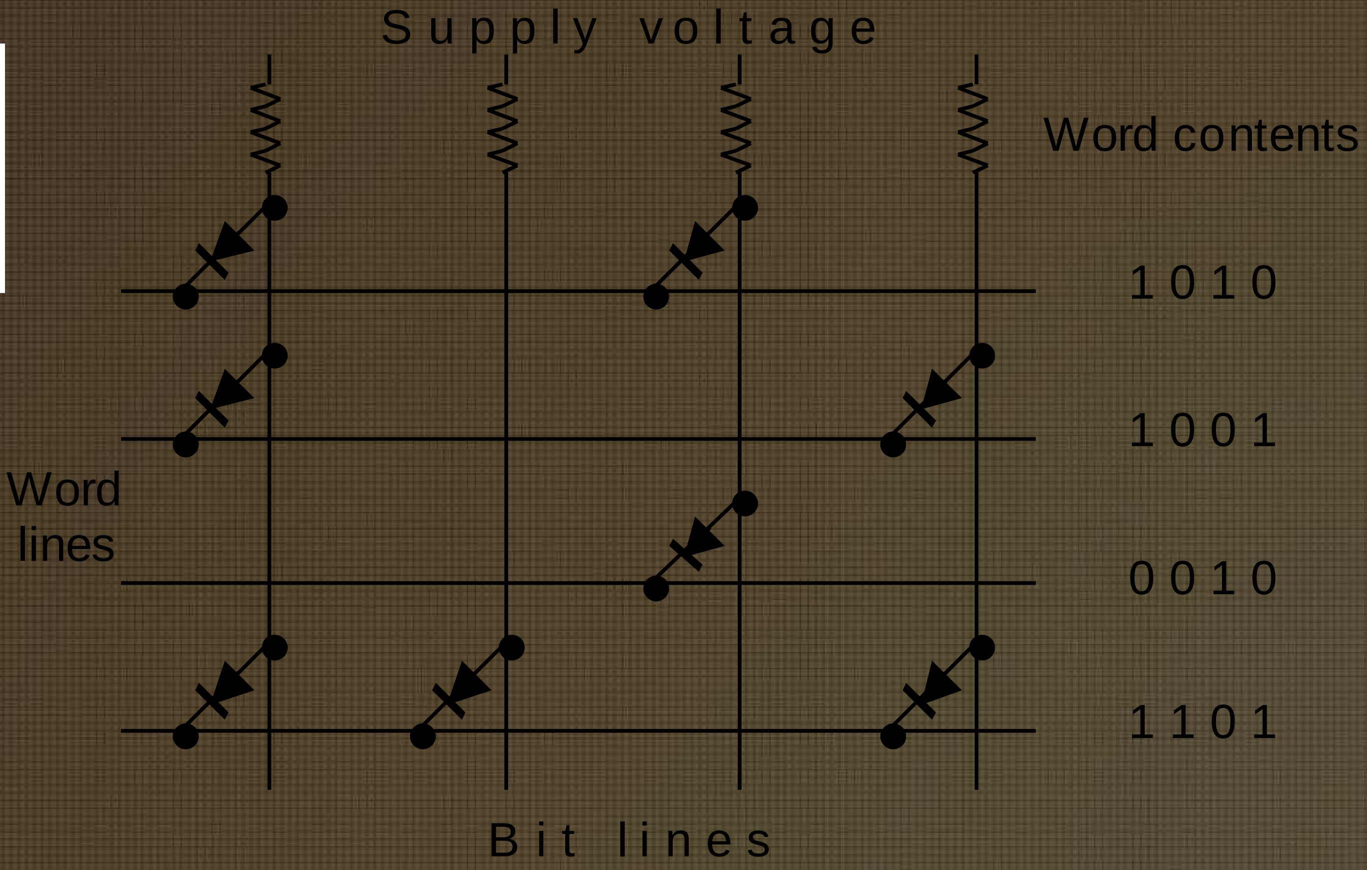

Wide-access memory
Mux
Narrow bus to processor
(a) Buffer and multiplexer at the memory side
Wide-access memory
Wide bus to processor
Mux
(a) Buffer and multiplexer at the processor side

Fig. 17.9 Two ways of using a wide-access memory to bridge the speed gap between the processor and memory.

ROM
PROM
EPROM



Read-only memory organization, with the fixed contents shown on the right.

The Need for a Memory Hierarchy

The widening speed gap between CPU and main memory

Processor operations take of the order of 1 ns

Memory access requires 10s or even 100s of ns

Memory bandwidth limits the instruction execution rate

Each instruction executed involves at least one memory access

Hence, a few to 100s of MIPS is the best that can be achieved

A fast buffer memory can help bridge the CPU-memory gap

The fastest memories are expensive and thus not very large

A second (third?) intermediate cache level is thus often used

Typical Levels in a Hierarchical Memory

Capacity	Access latency	Cost per GB
100s B	ns	$Millions
10s KB	a few ns	$100s Ks
MBs	10s ns	$10s Ks
100s MB	100s ns	$1000s
10s GB	10s ms	$10s
TBs	min+	$1s

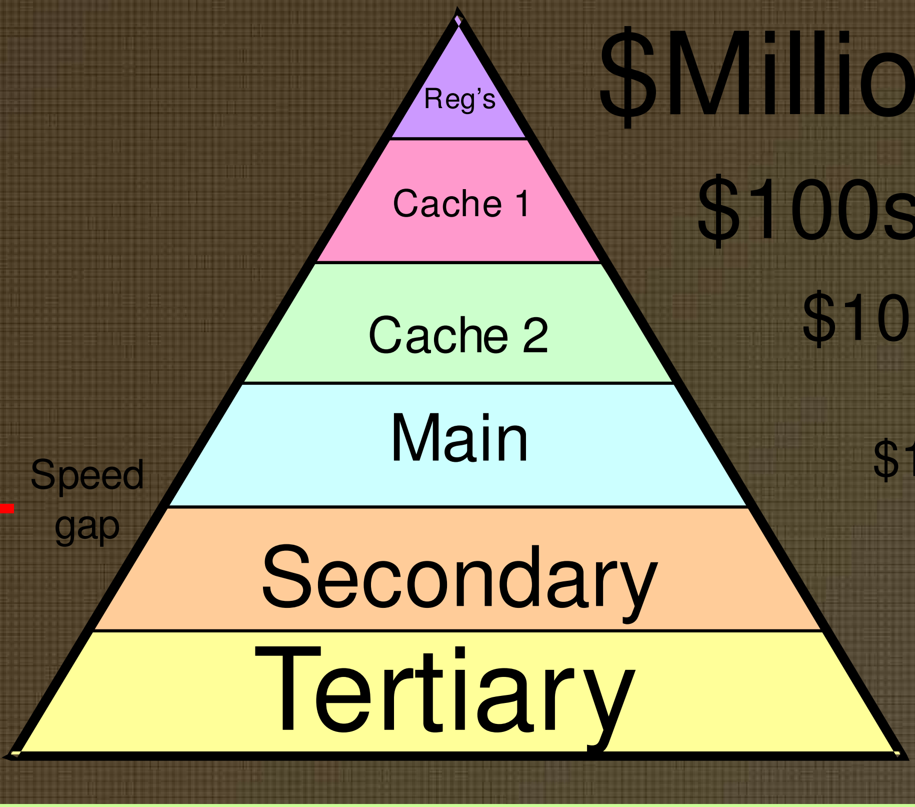


Fig. 17.14 Names and key characteristics of levels in a memory hierarchy.

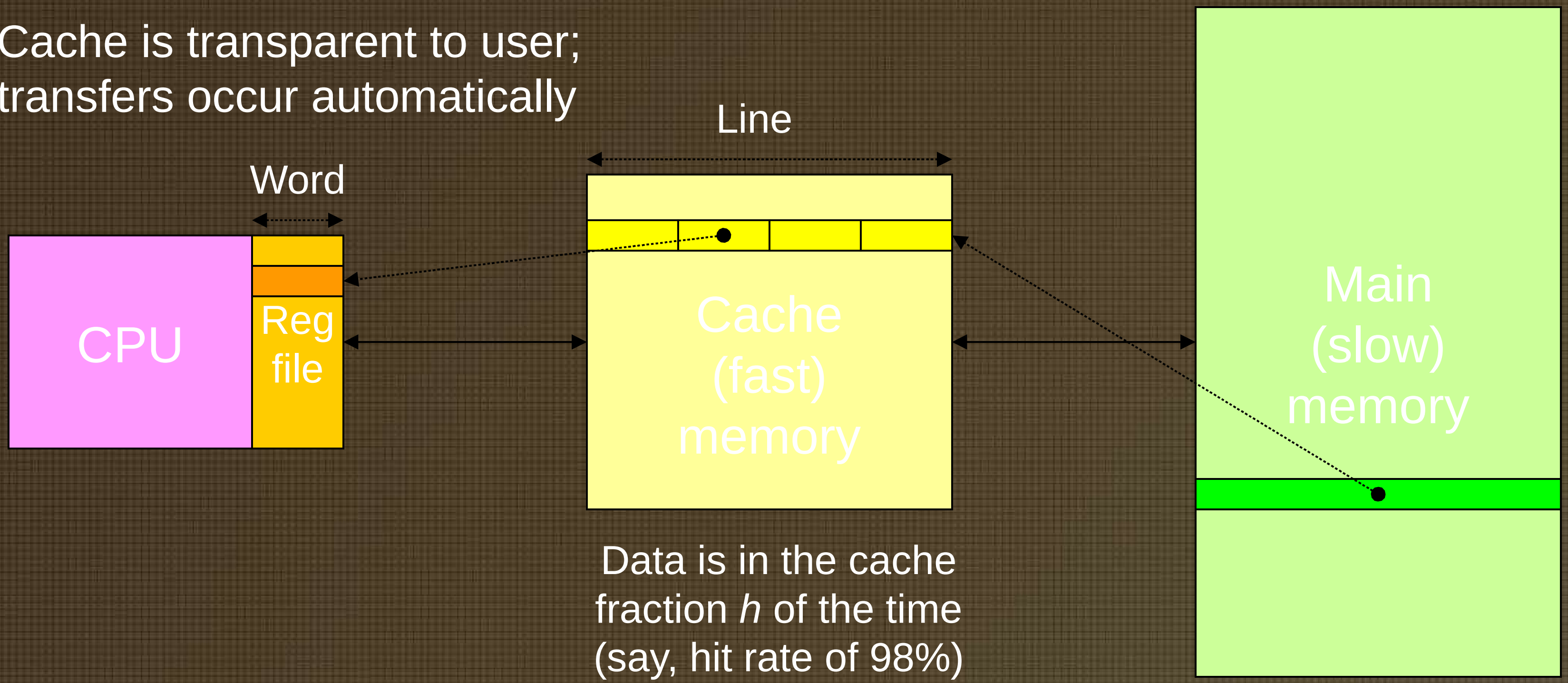
Cache Memory Organization

Processor speed is improving at a faster rate than memory’s

- Processor-memory speed gap has been widening
- Cache is to main as desk drawer is to file cabinet

Cache, Hit/Miss Rate, and Effective Access Time

Cache is transparent to user;
transfers occur automatically



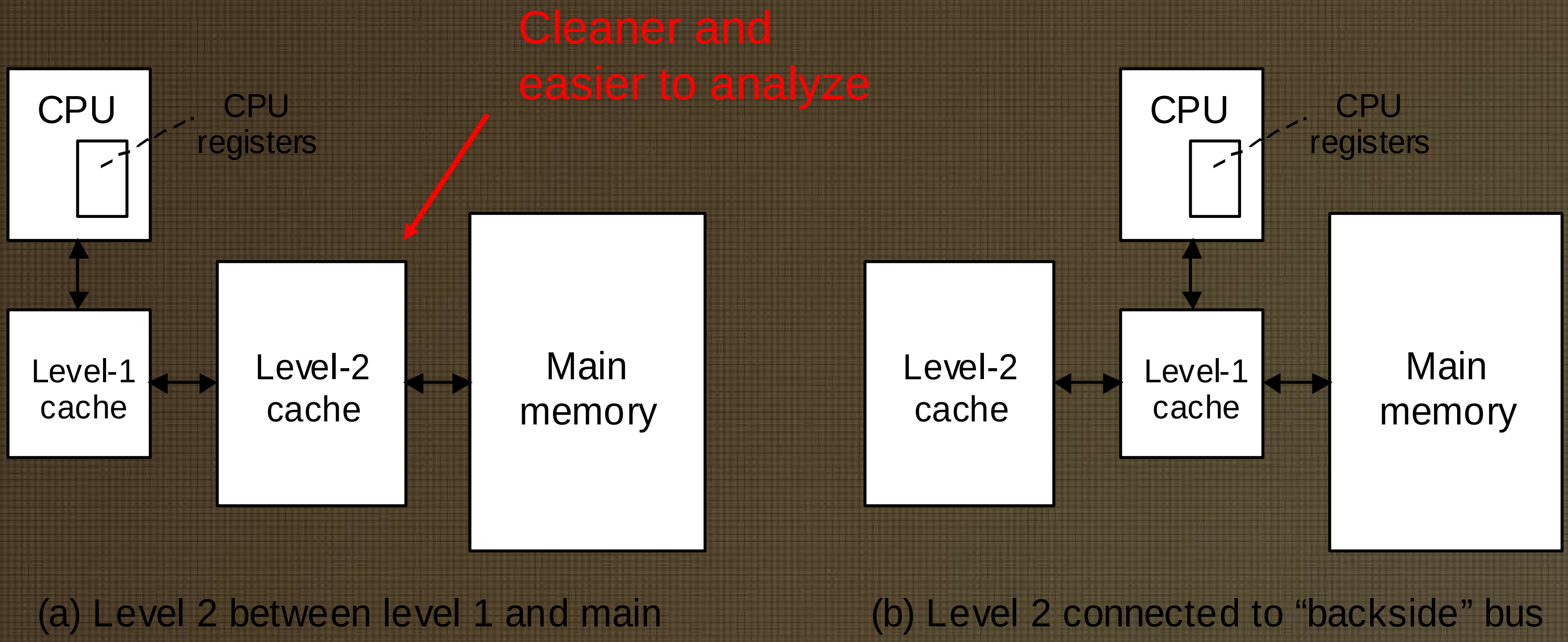
Data is in the cache
fraction *h* of the time
(say, hit rate of 98%)

Go to main 1 – *h* of the time
(say, cache miss rate of 2%)

One level of cache with hit rate *h*

$$C_{\text{eff}} = hC_{\text{fast}} + (1 - h)(C_{\text{slow}} + C_{\text{fast}}) = C_{\text{fast}} + (1 - h)C_{\text{slow}}$$

Multiple Cache Levels



(a) Level 2 between level 1 and main

(b) Level 2 connected to "backside" bus

Fig. 18.1 Cache memories act as intermediaries between the superfast processor and the much slower main memory.

Cache Memory Design Parameters

Cache size (in bytes or words). A larger cache can hold more of the program’s useful data but is more costly and likely to be slower.

Block or *cache-line size* (unit of data transfer between cache and main). With a larger cache line, more data is brought in cache with each miss. This can improve the hit rate but also may bring low-utility data in.

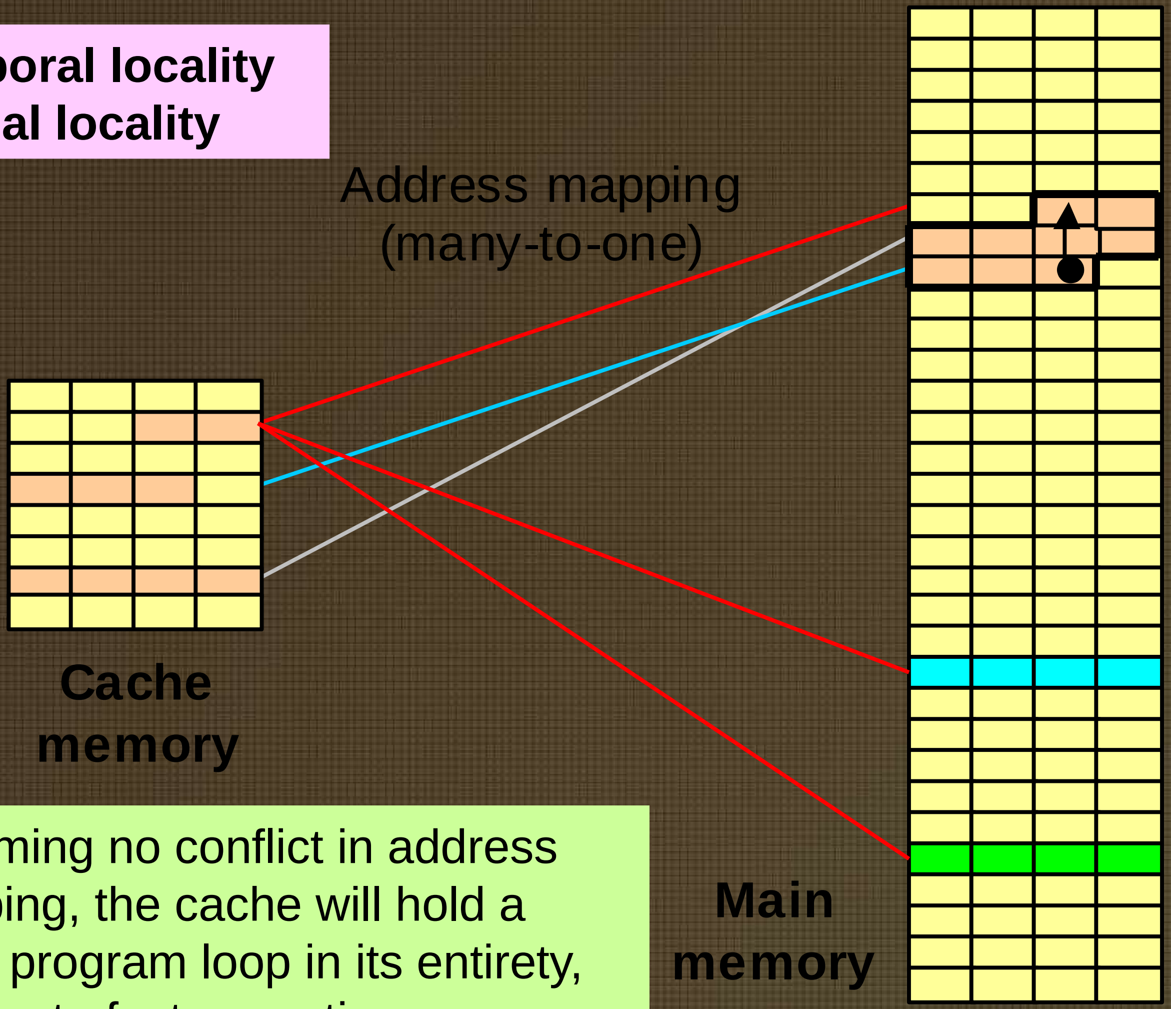
Placement policy. Determining where an incoming cache line is stored. More flexible policies imply higher hardware cost and may or may not have performance benefits (due to more complex data location).

Replacement policy. Determining which of several existing cache blocks (into which a new cache line can be mapped) should be overwritten. Typical policies: choosing a random or the least recently used block.

Write policy. Determining if updates to cache words are immediately forwarded to main (*write-through*) or modified blocks are copied back to main if and when they must be replaced (*write-back* or *copy-back*).

What Makes a Cache Work?

Temporal locality
Spatial locality



Assuming no conflict in address mapping, the cache will hold a small program loop in its entirety, leading to fast execution.

Direct-Mapped Cache

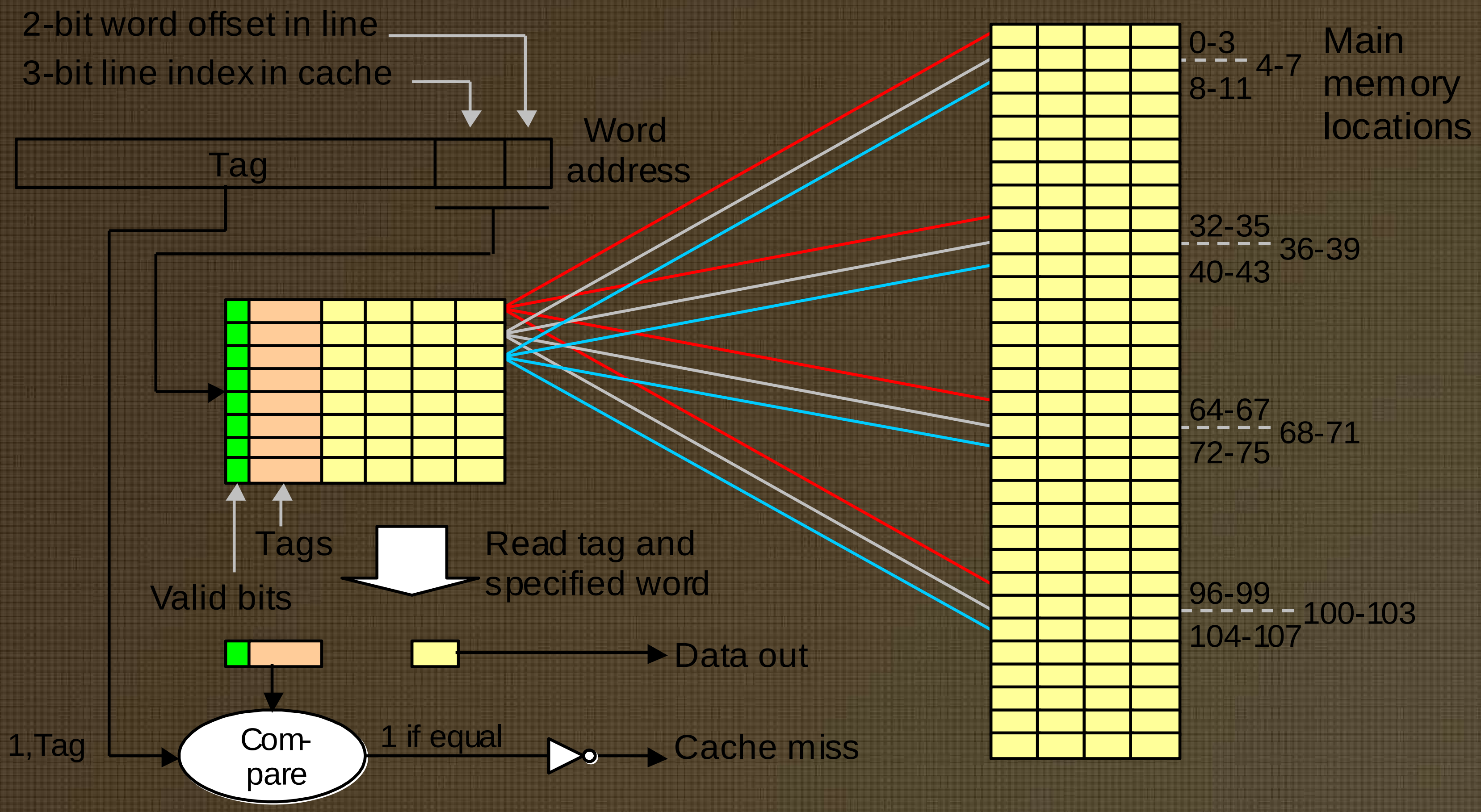


Fig. 18.4 Direct-mapped cache holding 32 words within eight 4-word lines. Each line is associated with a tag and a valid bit.

Accessing a Direct-Mapped Cache

Example

Show cache addressing for a byte-addressable memory with 32-bit addresses. Cache line *W* = 16 B. Cache size *L* = 4096 lines (64 KB).

Solution

Byte offset in line is $\log_2 16 = 4$ b. Cache line index is $\log_2 4096 = 12$ b.

This leaves 32 – 12 – 4 = 16 b for the tag.

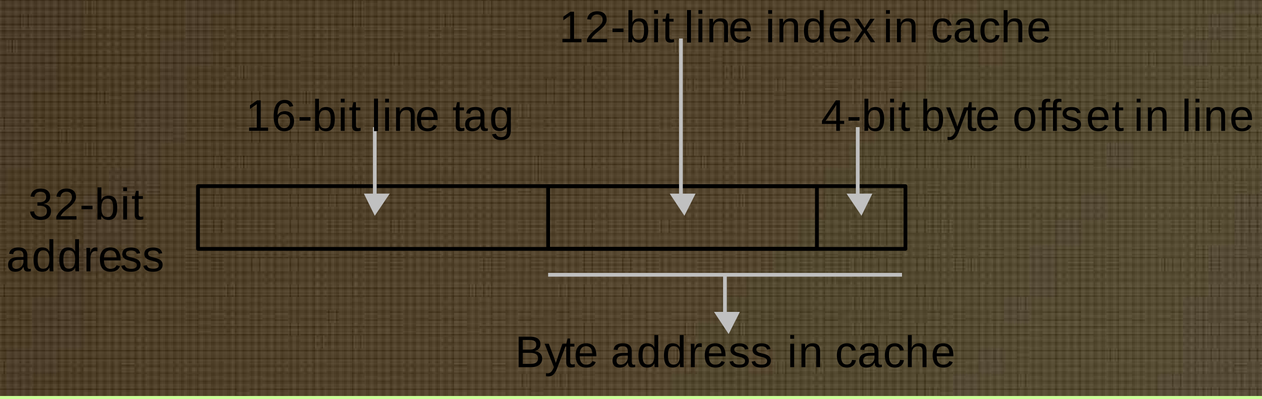
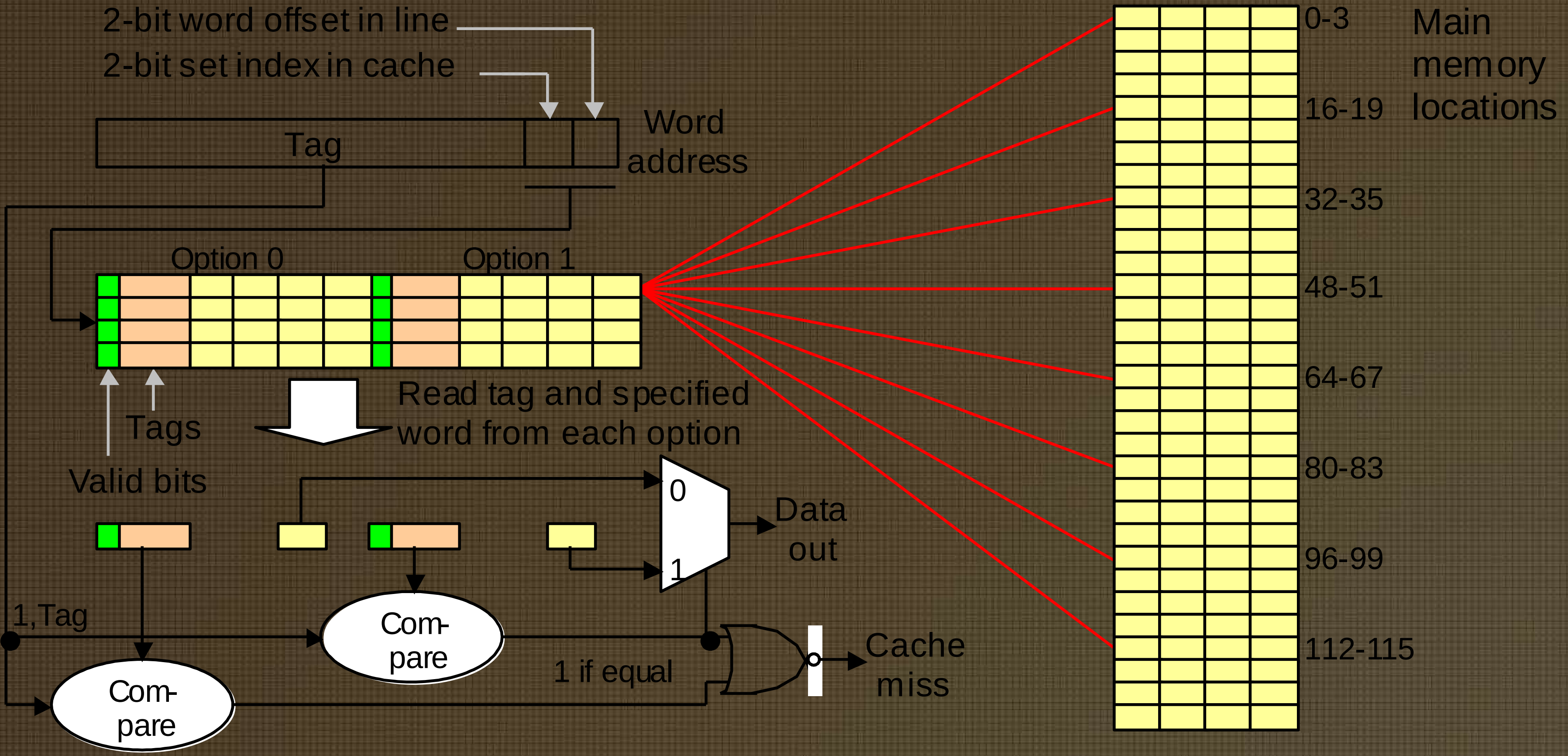


Fig. 18.5 Components of the 32-bit address in an example direct-mapped cache with byte addressing.

Set-Associative Cache



Two-way set-associative cache holding 32 words of data within 4-word lines and 2-line sets.

Accessing a Set-Associative Cache

Example

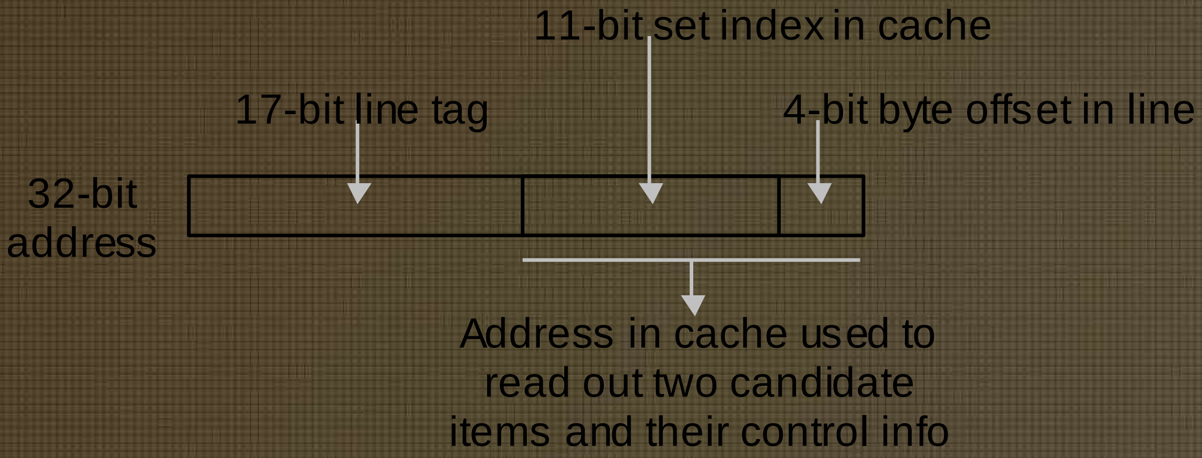
Show cache addressing scheme for a byte-addressable memory with 32-bit addresses. Cache line width $2^W = 16$ B. Set size $2^S = 2$ lines. Cache size $2^L = 4096$ lines (64 KB).

Solution

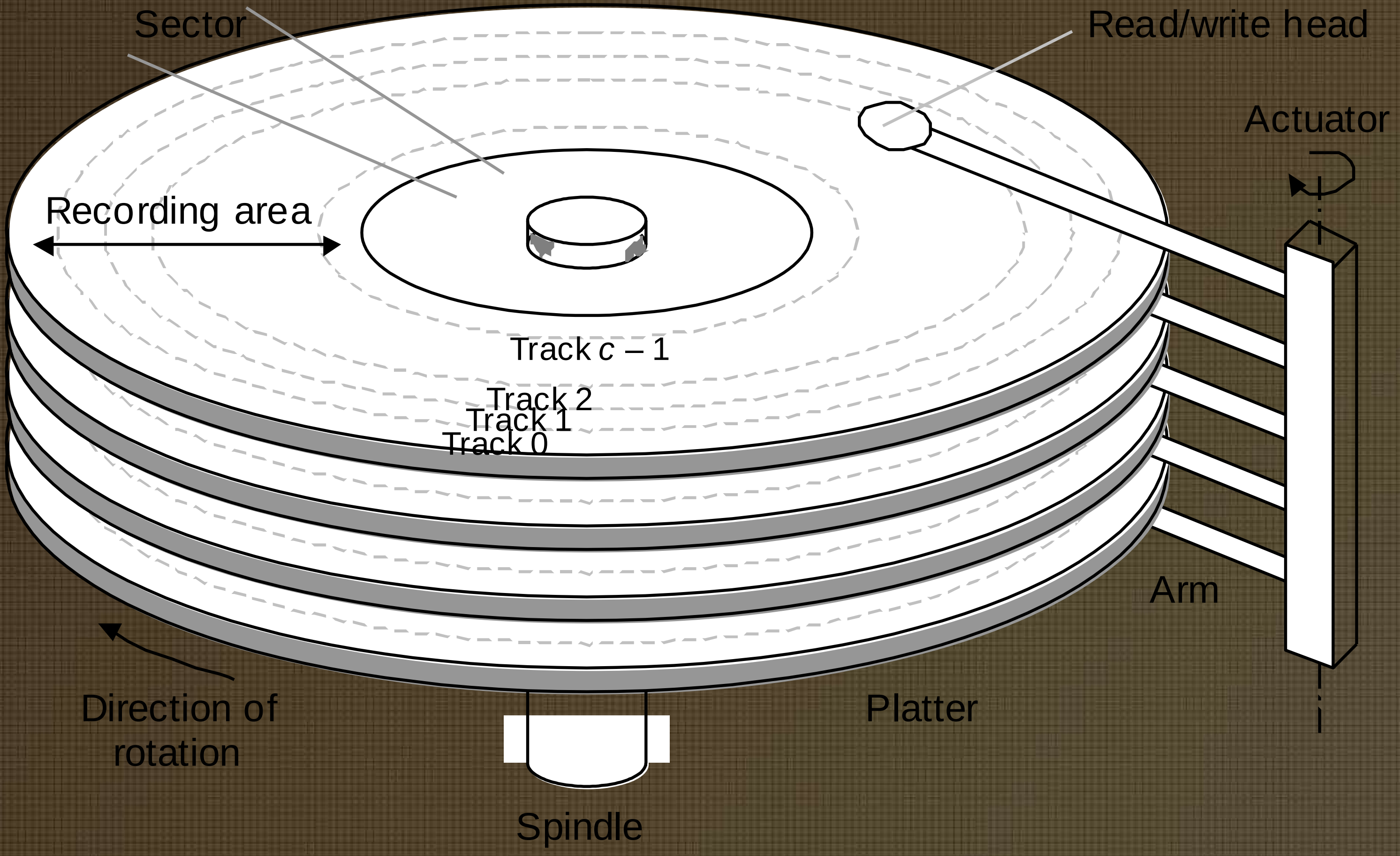
Byte offset in line is $\log_2 16 = 4$ b. Cache set index is $(\log_2 4096/2) = 11$ b.

This leaves 32 – 11 – 4 = 17 b for the tag.

Components of the 32-bit address in an example two-way set-associative cache.

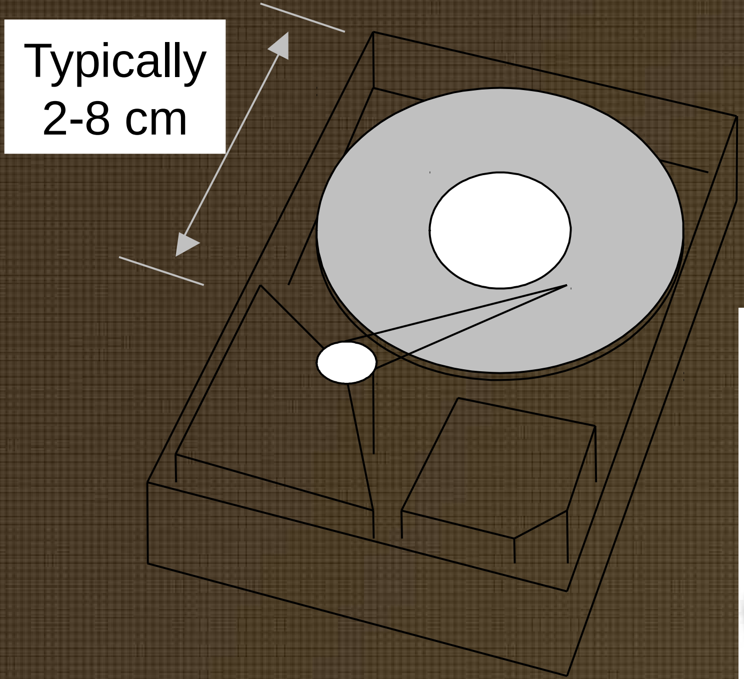

11-bit set index in cache
17-bit line tag
4-bit byte offset in line
32-bit address
Address in cache used to read out two candidate items and their control info

Disk Memory Basics

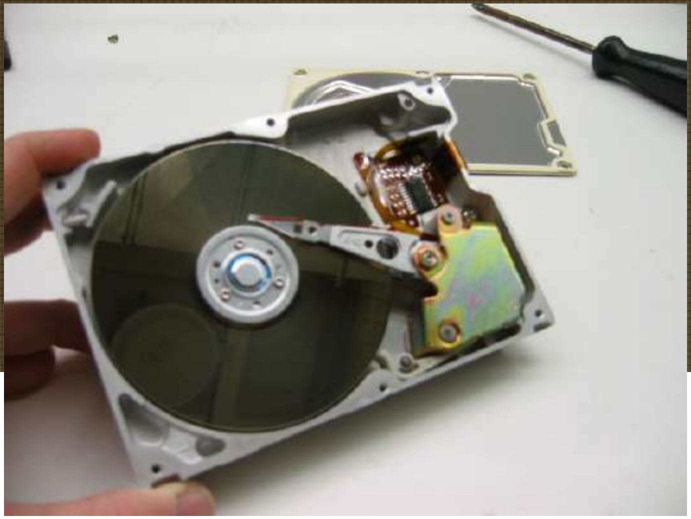


Disk memory elements and key terms.

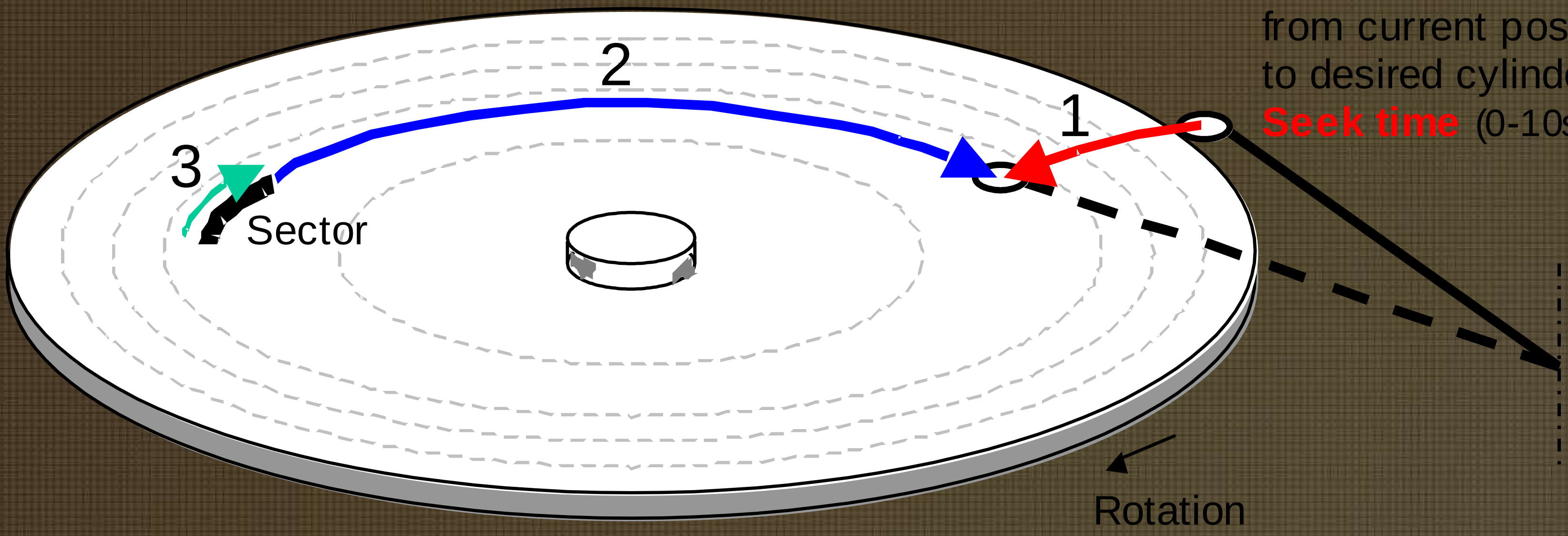
Disk Drives



Typically 2-8 cm



Access Time for a Disk



3. Disk rotation until sector has passed under the head: **Data transfer time** (< 1 ms)

2. Disk rotation until the desired sector arrives under the head: **Rotational latency** (0-10s ms)

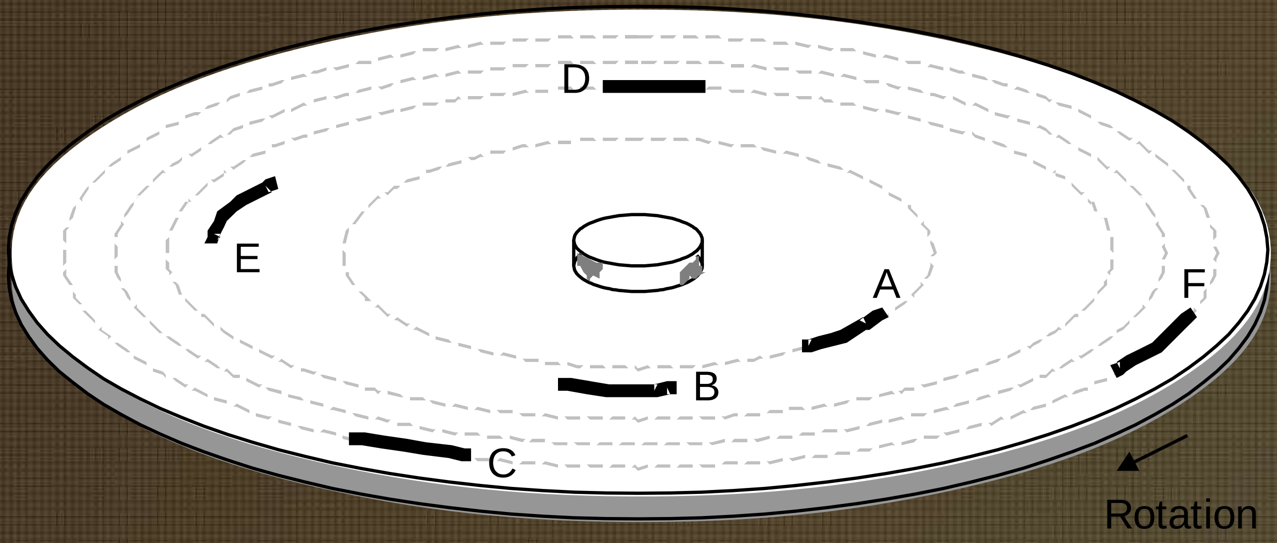
1. Head movement from current position to desired cylinder: **Seek time** (0-10s ms)

The three components of disk access time. Disks that spin faster have a shorter average and worst-case access time.

Disk Performance

Seek time = $a + b(c - 1) + \beta(c - 1)^{1/2}$

Average rotational latency = (30 / rpm) s = (30 000 / rpm) ms



Arrival order of access requests:

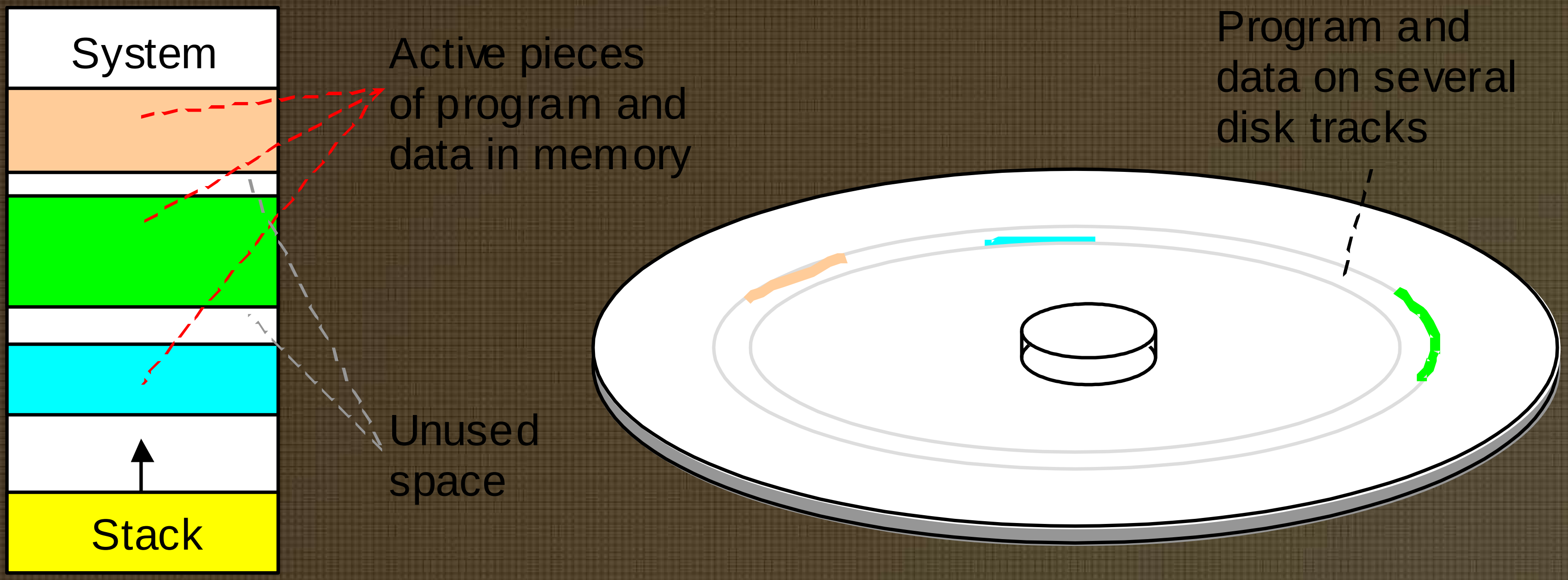
A, B, C, D, E, F

Possible out-of-order reading:

C, F, D, E, B, A

Reducing average seek time and rotational latency by performing disk accesses out of order.

The Need for Virtual Memory


System
Active pieces of program and data in memory
Program and data on several disk tracks
Unused space
Stack

Program segments in main memory and on disk.

Memory Hierarchy: The Big Picture

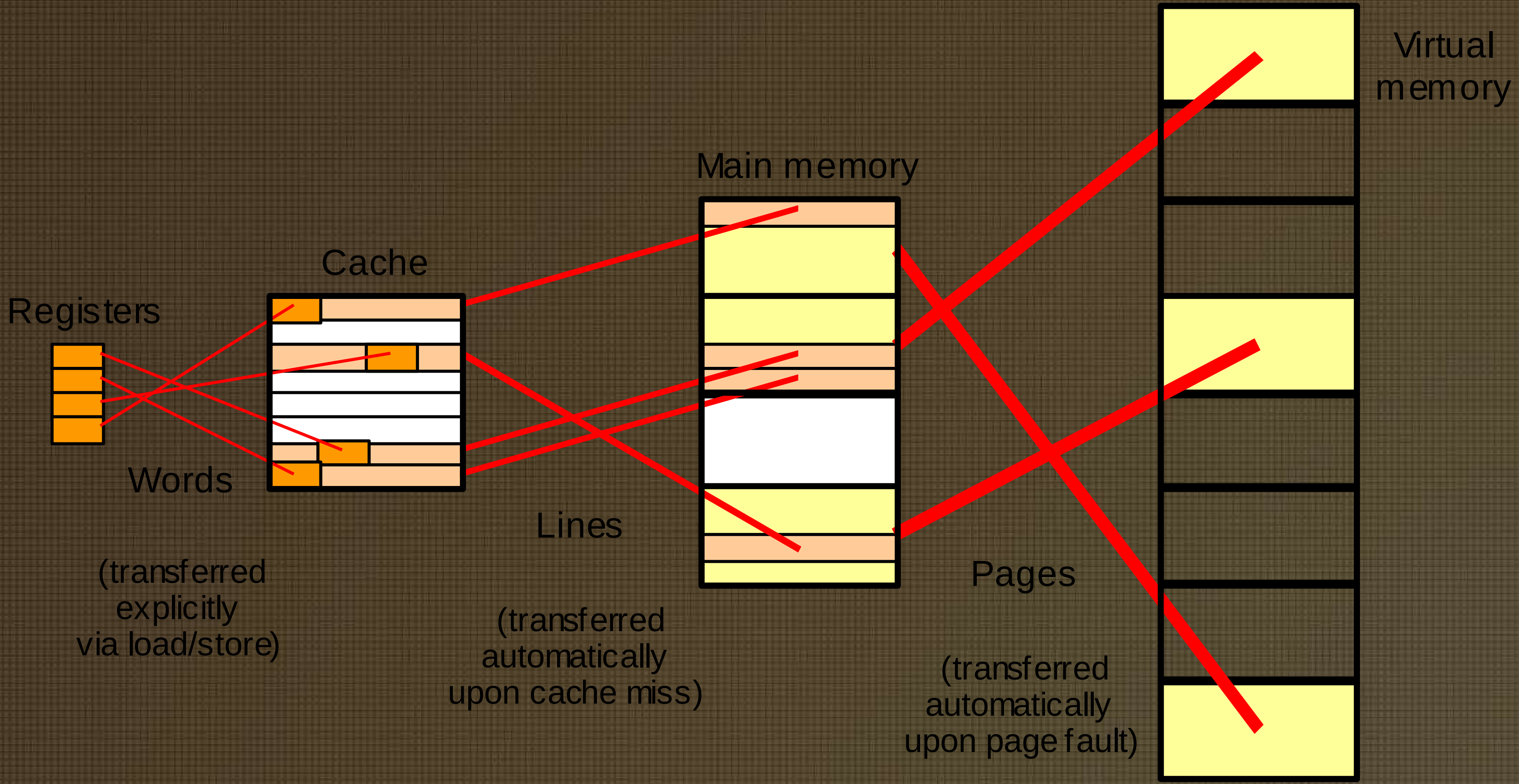

Virtual memory
Main memory
Cache
Registers
Words
(transferred explicitly via load/store)
Lines
(transferred automatically upon cache miss)
Pages
(transferred automatically upon page fault)

Fig. Data movement in a memory hierarchy.

Address Translation in Virtual Memory

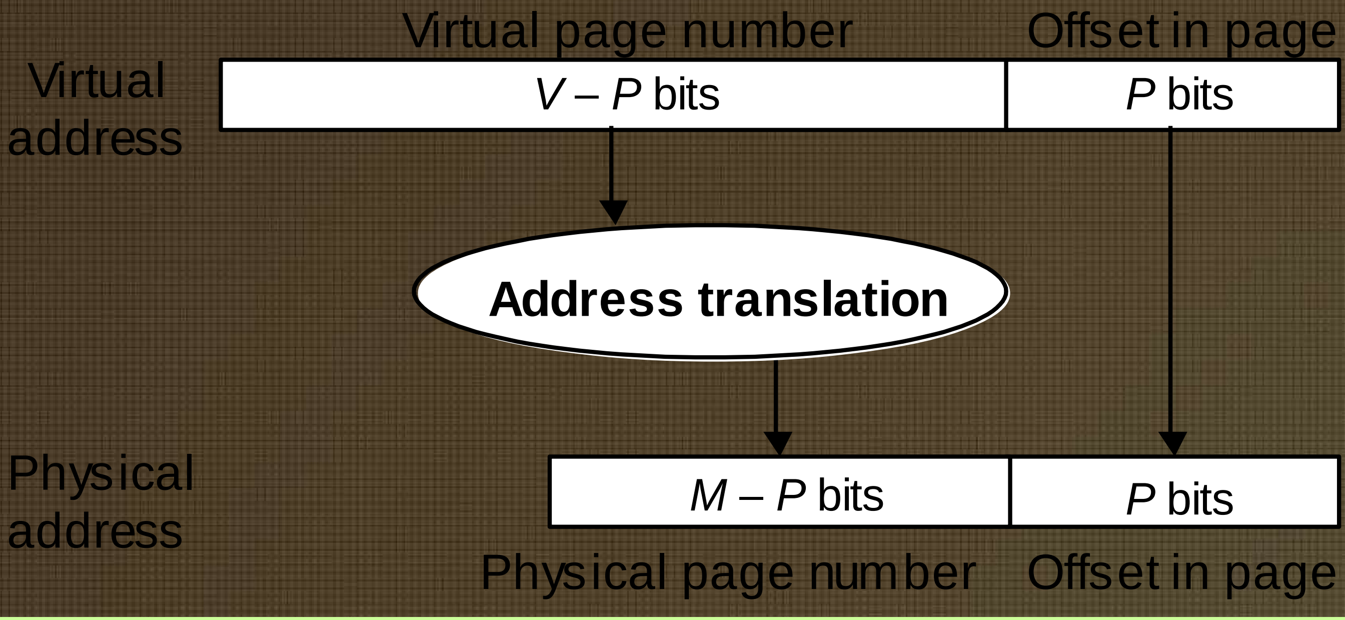


Fig.Virtual-to-physical address translation parameters.

Example

Determine the parameters in Fig. 20.3 for 32-bit virtual addresses, 4 KB pages, and 128 MB byte-addressable main memory.

Solution: Physical addresses are 27 b, byte offset in page is 12 b; thus, virtual (physical) page numbers are 32 – 12 = 20 b (15 b)

Page Tables and Address Translation

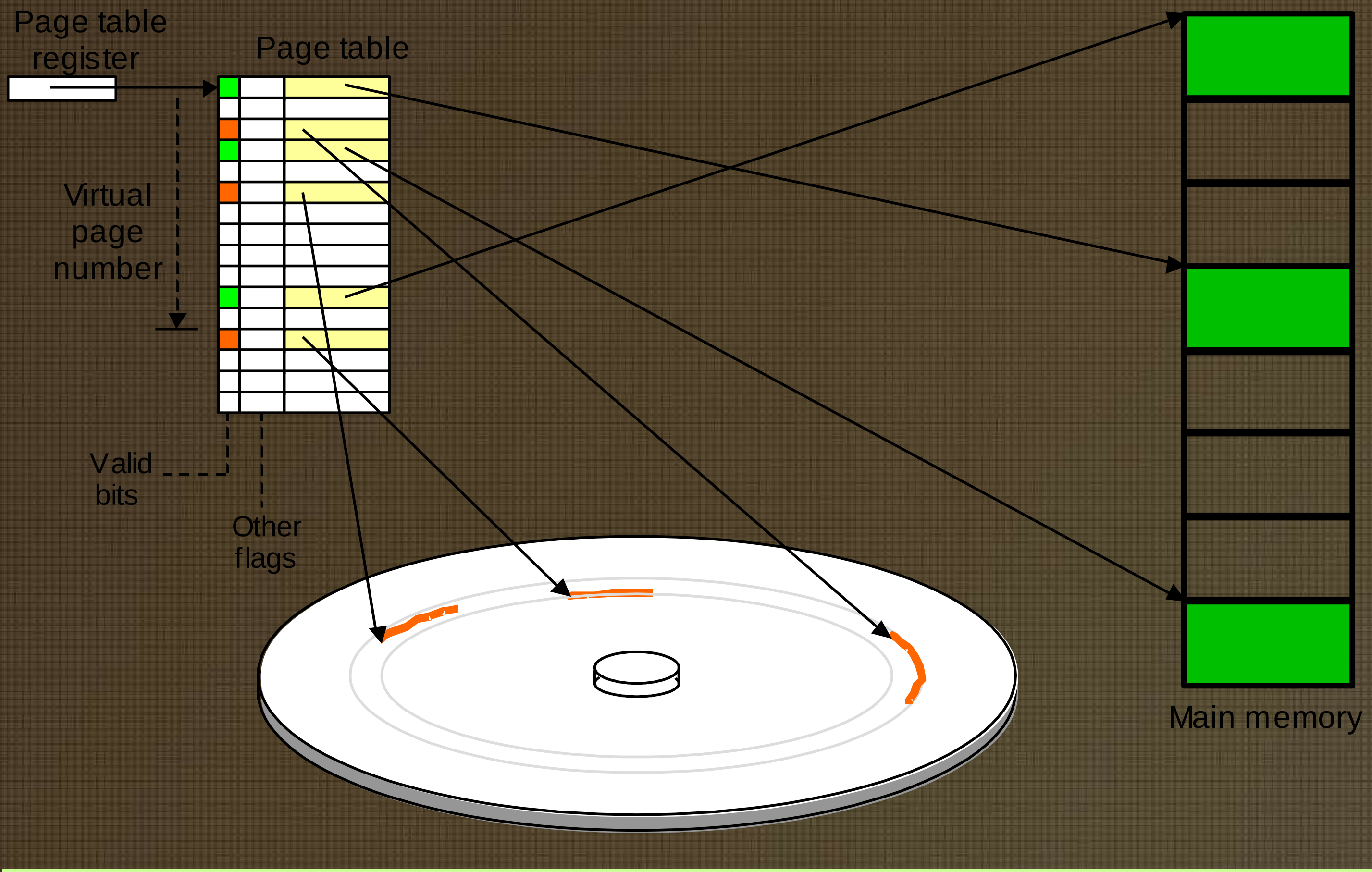


Fig. The role of page table in the virtual-to-physical address translation process.

Protection and Sharing in Virtual Memory

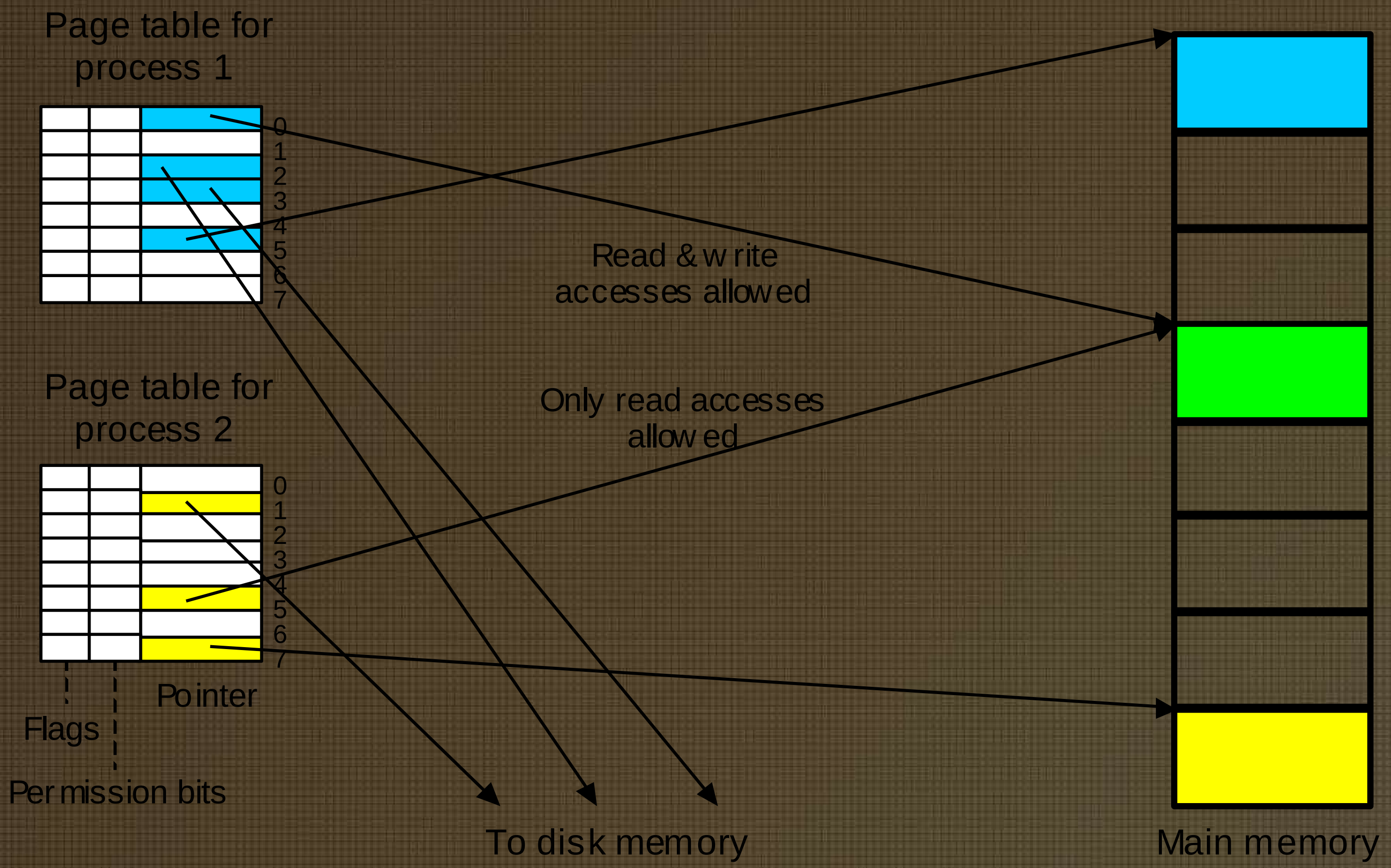


Fig. Virtual memory as a facilitator of sharing and memory protection.

The Latency Penalty of Virtual Memory

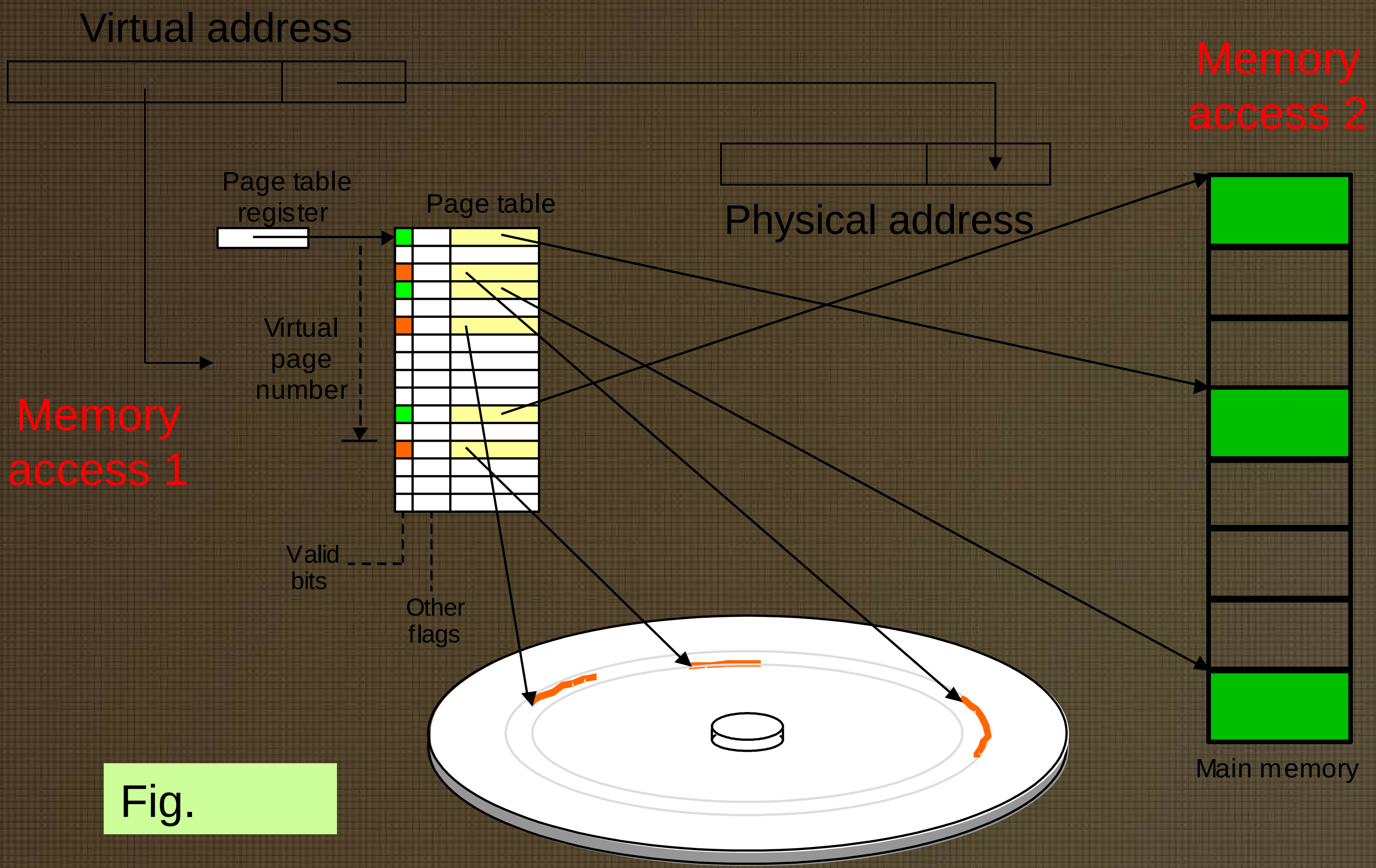
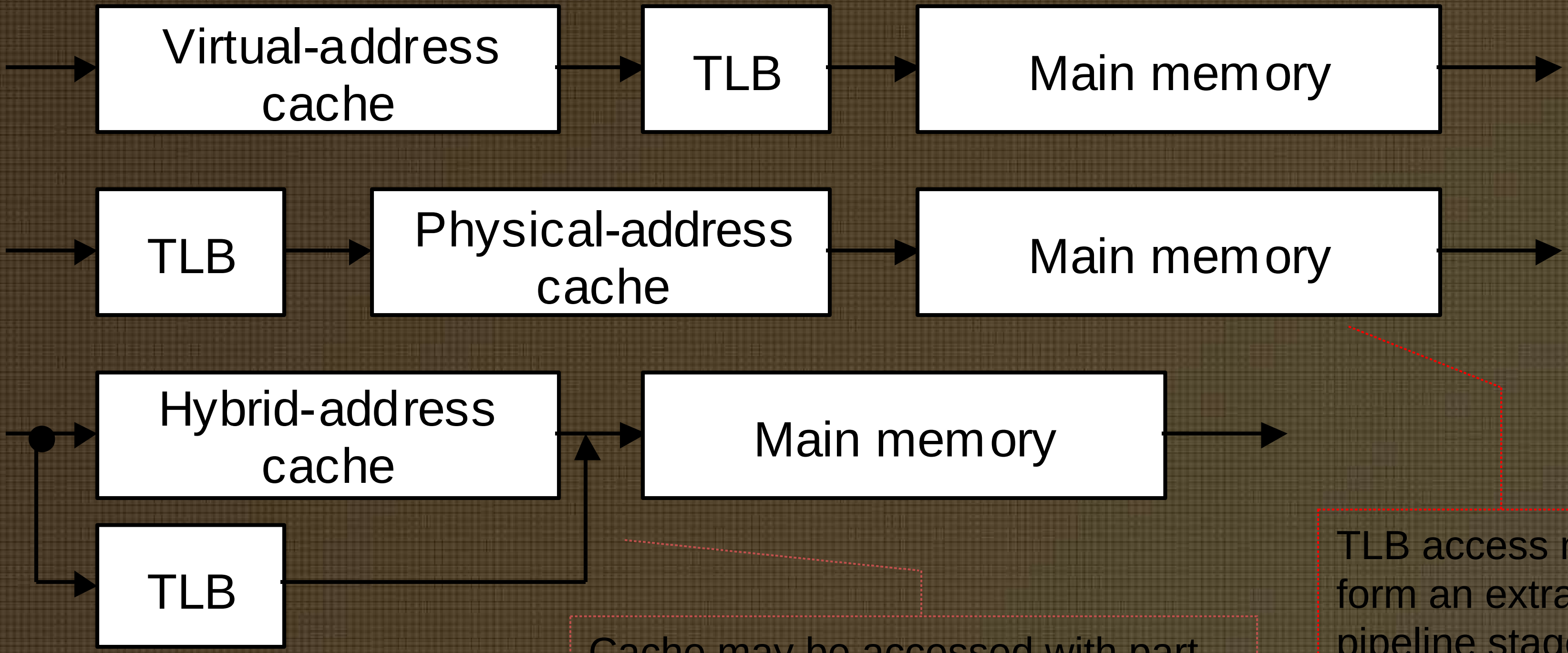


Fig.

Virtual- or Physical-Address Cache?



Cache may be accessed with part of address that is common between virtual and physical addresses

TLB access may form an extra pipeline stage, thus the penalty in throughput can be insignificant

Fig.Options for where virtual-to-physical address translation occurs.

Least-recently used (LRU) policy

Implemented by maintaining a stack

Pages A B A F B E A

LRU stack

MRU		D	A	B	A	F	B	E	A
	B	D	A	B	A	F	B	E	
	E	B	D	D	B	A	F	B	
LRU		C	E	E	E	D	D	A	F

Approximate LRU Replacement Policy

Least-recently used policy: effective, but hard to implement

Approximate versions of LRU are more easily implemented

- Clock policy: diagram below shows the reason for name
- Use bit is set to 1 whenever a page is accessed

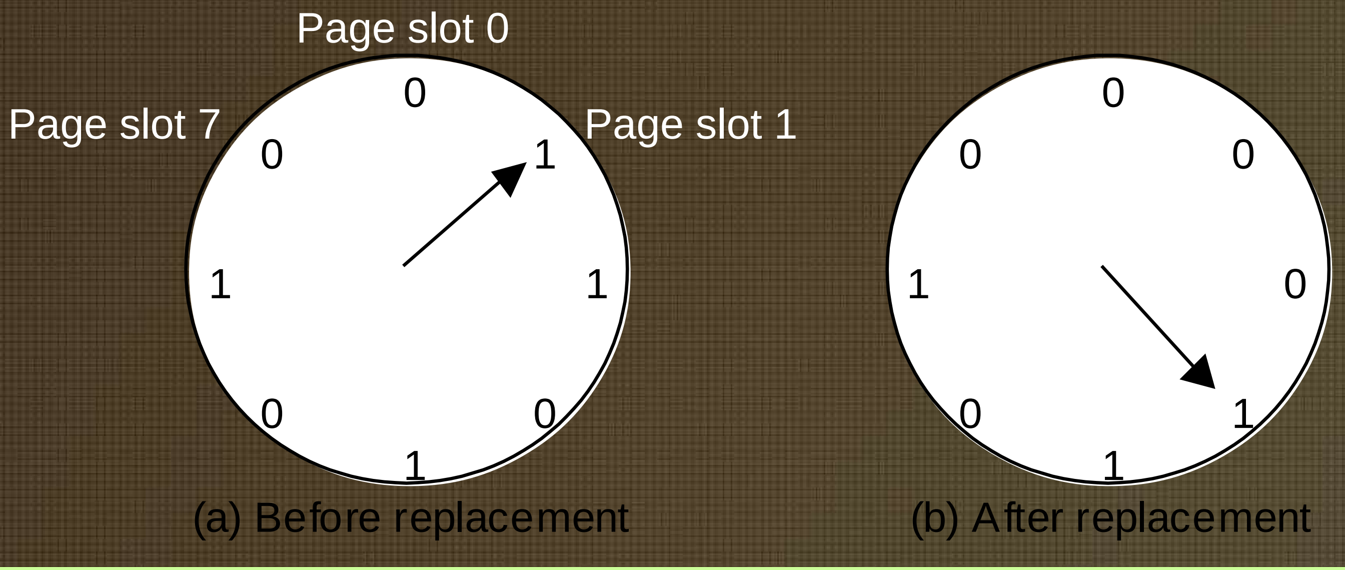


Fig. 20.8 A scheme for the approximate implementation of LRU .

Improving Virtual Memory Performance

Table 20.1 Memory hierarchy parameters and their effects on performance

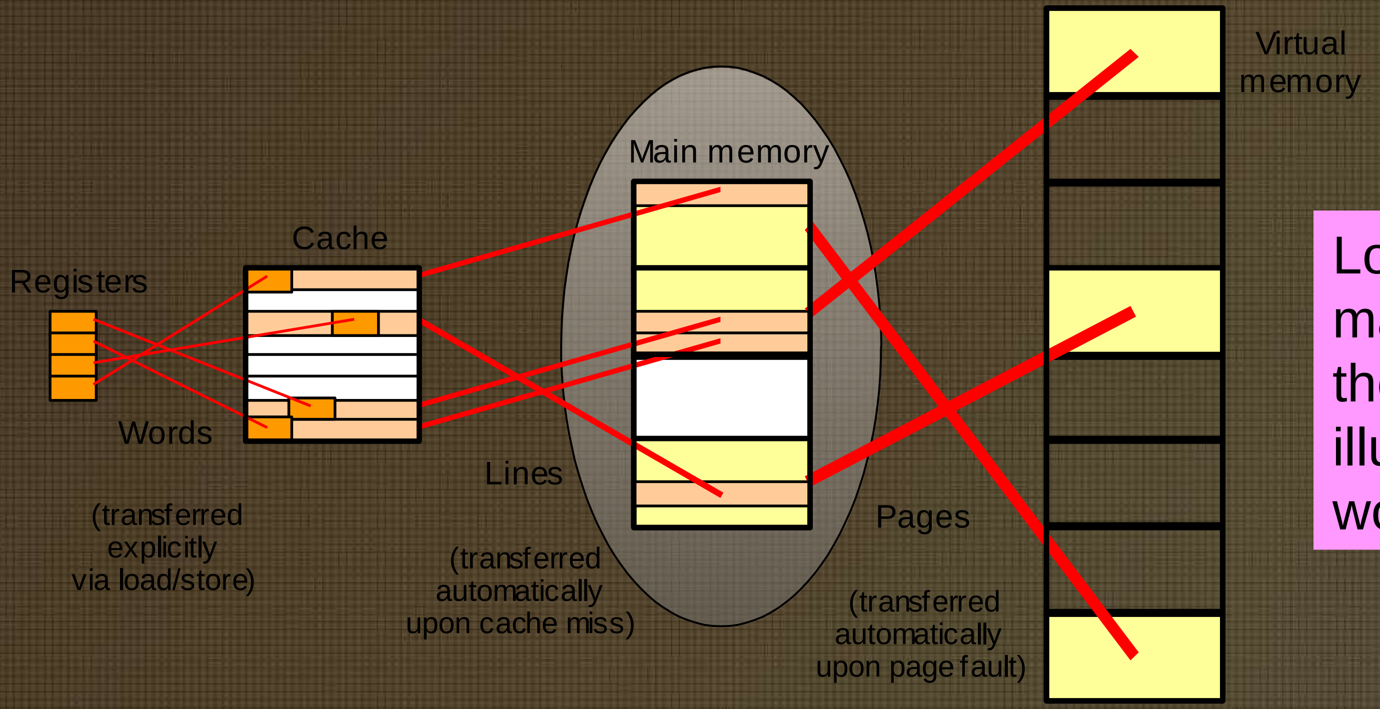
Parameter variation	Potential advantages	Possible disadvantages
Larger main or cache size	Fewer capacity misses	Longer access time
Larger pages or longer lines	Fewer compulsory misses (prefetching effect)	Greater miss penalty
Greater associativity (for cache only)	Fewer conflict misses	Longer access time
More sophisticated replacement policy	Fewer conflict misses	Longer decision time, more hardware
Write-through policy (for cache only)	No write-back time penalty, easier write-miss handling	Wasted memory bandwidth, longer access time

Summary of Memory Hierarchy

Cache memory: provides illusion of very high speed

Main memory: reasonable cost, but slow & small

Virtual memory: provides illusion of very large size


Virtual memory
Main memory
Cache
Registers
Words
Lines
Pages
(transferred explicitly via load/store)
(transferred automatically upon cache miss)
(transferred automatically upon page fault)

Locality makes the illusions work

Fig. Data movement in a memory hierarchy.

Assignment 3

Q1. What do you mean by memory hierarchy?

Q2.Discuss semiconductor RAM. Differentiate SRAM and DRAM.

Q3. What is cache memory? Why it is implemented?

Q4.Explain the concept of virtual memory.

Q5. Write short notes on following:

i. Optical Disk

ii. Magnetic Tape

Tutorial 3

Q1. A computer uses RAM chip of 1024*1 capacity. How many chips are needed and how should there address lines be connected to provide a memory capacity of 1024 bytes?

Q2.A ROM chip of 1024*8 bits has four select inputs and operates on a 5 volt power supply. How many pins are needed for the IC package? Draw a block diagram and label all inputs and output terminals in the ROM.

Q3. Explain various cache mapping techniques. A computer system has 4K word cache organized in block set associative manner with 4 blocks per set, 64 words per block. The main memory contains 65536 blocks. How many bits are there in each of TAG, SET and WORD fields?

Q4. Define the term address space and memory space. An address space is specified by 24 bits and corresponding memory space specified by 16 bits. Find the following:

i. How many words are there in address space?

ii. How many words are there in memory space?

iii. If a page consists of 2K word, how many pages and blocks are there in the system?

Q5. A Computer employs RAM chips of 2568 and ROM chips of 1024*8. The computer needs 2KB of RAM and

4KB of ROM and 4 interface units, each of 4 registers:

i. How many RAM and ROM chips are needed

ii. Draw memory address map for the system.

Outcomes

After reading above topics students will be able to:

- Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

COMPUTER ORGANIZATION & ARCHITECTURE

UNIT 4

I/O Organization

Presentation Outline

- Peripheral Devices
- I/O
  - Modules
  - Interfaces
  - Processor
  - Port
- Modes of Transfer
  - Programmed I/O
  - Interrupt Driven I/O
  - DMA
- Asynchronous & Synchronous Communication

Learning Objectives

The objectives of the following slide is to make student aware about the :

- Peripheral Devices
- I/O
  - Modules
  - Interfaces
  - Processor
  - Port
- Modes of Transfer
  - Programmed I/O
  - Interrupt Driven I/O
  - DMA
- Asynchronous & Synchronous Communication

Input-Output Organization

- 11-1 Peripheral Devices
  - I/O Subsystem
    - Provides an efficient mode of communication between the central system and the outside environment
  - Peripheral (or I/O Device)
    - Input or Output devices attached to the computer
      - Monitor (*Visual Output Device*) : CRT, LCD
      - KBD (*Input Device*) : light pen, mouse, touch screen, joy stick, digitizer
      - Printer (Hard Copy Device) : Dot matrix (*impact*), thermal, ink jet, laser (*non-impact*)
      - Storage Device : Magnetic tape, magnetic disk, optical disk
  - ASCII (*American Standard Code for Information Interchange*) Alphanumeric Characters
    - I/O communications are usually involved in the transfer of ASCII information
    - ASCII Code
    - 11-2 Input-Output Interface
  - Interface
    - 1) A conversion of signal values may be required

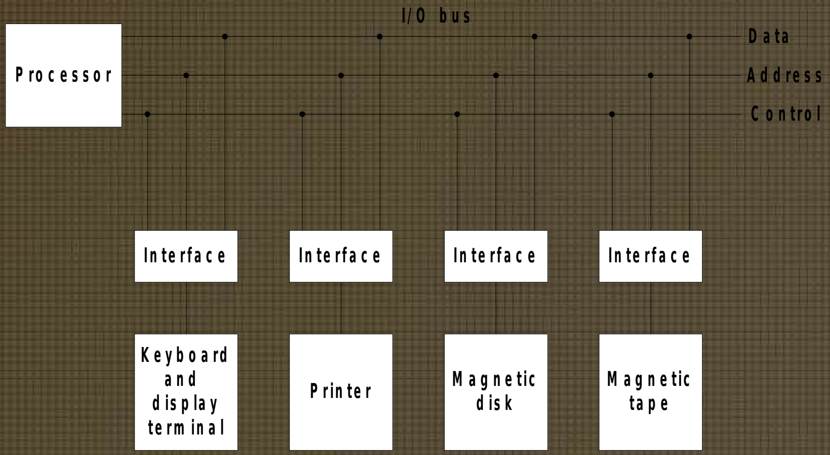
- 2) A synchronization mechanism may be needed
  - The data transfer rate of peripherals is usually slower than the transfer rate of the CPU
- 3) Data codes and formats in peripherals differ from the word format in the CPU and Memory
- 4) The operating modes of peripherals are different from each other
  - Each peripherals must be controlled so as not to disturb the operation of other peripherals connected to the CPU

– Interface

- Special hardware components between the CPU and peripherals
- Supervise and Synchronize all input and output transfers

– I/O Bus and Interface Modules : ***Fig. 11-1***

- I/O Bus
  - Data lines
  - Address lines
  - Control lines
- Interface Modules : *주로 VLSI Chip 사용*
  - SCSI (Small Computer System Interface)
  - IDE (Integrated Device Electronics)
  - Centronics
  - RS-232
  - IEEE-488 (GPIB)



- I/O command : 8251 SIO
  - Control Command
  - Status Command
  - Input Command
  - Output Command
- I/O Bus versus Memory Bus
  - Computer buses can be used to communicate with memory and I/O

Intel, Zilog

  - 1) Use two separate buses, one for memory and the other for I/O : ***Fig. 11-19***, p. 421
    - » I/O Processor
  - 2) Use one common bus for both memory and I/O but have separate control lines for each : ***Isolated I/O*** or ***I/O Mapped I/O***
    - » **IN**, **OUT** : I/O Instruction
    - » **MO**

*** Control Lines**

I/O Request, Mem Request, Read/Write

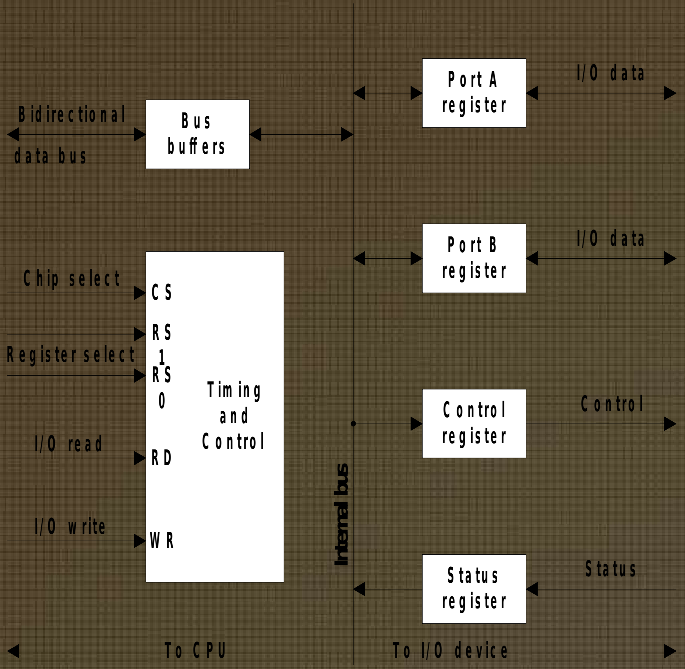
Motorola

  - 3) Use one common bus for memory and I/O with common control lines : ***Memory Mapped I/O***
    - » **MO** struction

*** Control Lines**

Read/Write

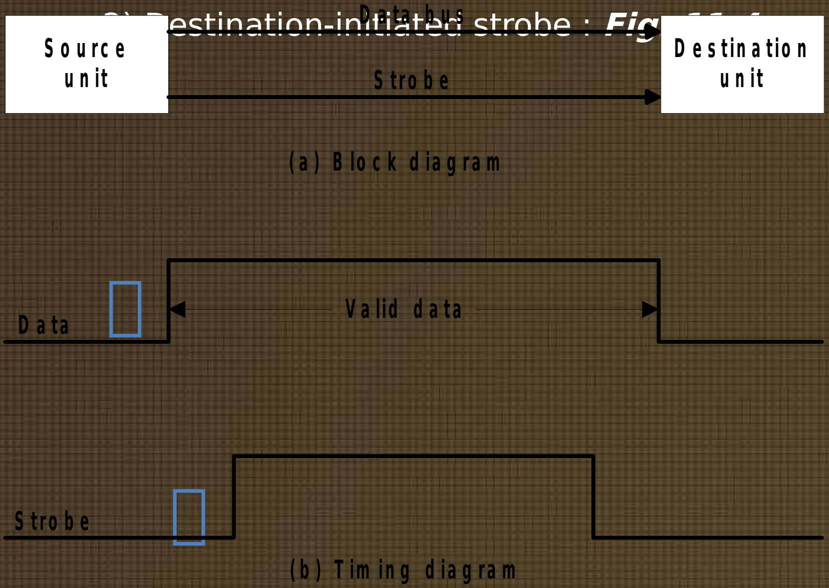
- Example of I/O Interface : ***Fig. 11-2***
  - 4 I/O port : Data port A, Data port B, Control, Status
    - 비교 : 8255 PIO (port A, B, C, Control/Status)
  - Address Decode : CS, RS1, RS0



CS	RS1	RS0	Register selected
0	x	x	None : data bus in high-impedance
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

- 11-3 Asynchronous Data Transfer
  - Synchronous Data Transfer
    - All data transfers occur simultaneously during the occurrence of a clock pulse
    - Registers in the **interface** share a common clock with **CPU** registers
  - Asynchronous Data Transfer
    - Internal timing in each unit (*CPU and Interface*) is independent
    - Each unit uses its own private clock for internal registers

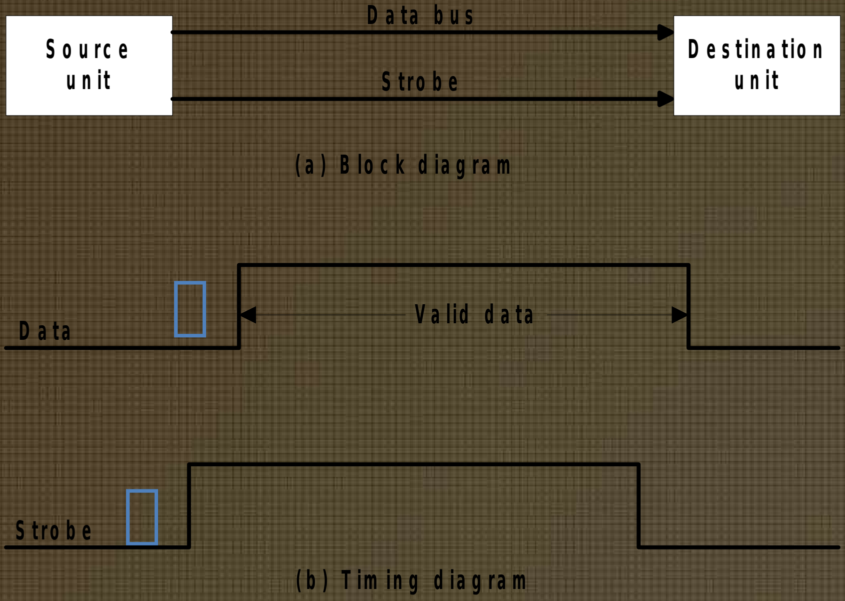
- Strobe : Control signal to indicate the time at which data is being transmitted
  - 1) Source-initiated strobe : ***Fig. 11-3***
  - 2) Destination-initiated strobe : ***Fig. 11-4***



(a) Block diagram

(b) Timing diagram

Fig. 11-3 Source-initiated strobe



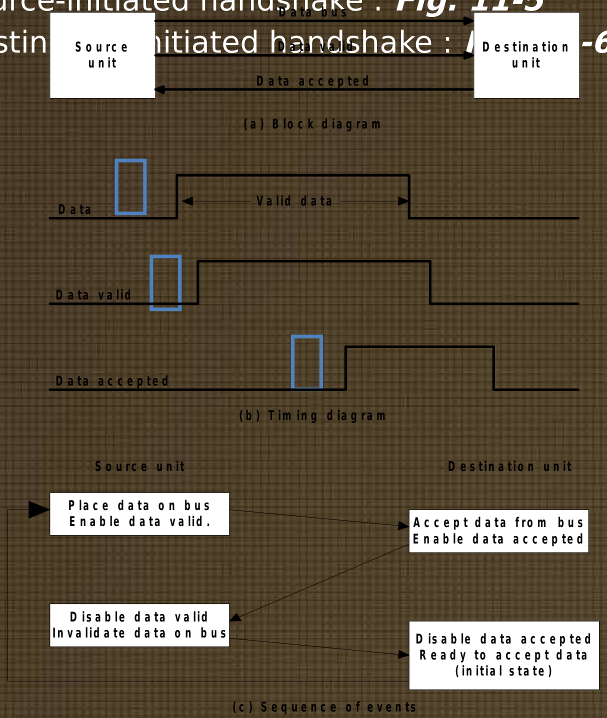
(a) Block diagram

(b) Timing diagram

Fig. 11-4 Destination-initiated strobe

- Disadvantage of strobe method
  - Destination 이 Data를 아무 이상 없이 잘 가져 가ㅇ는지 알 수가 없다.
  - 따라서 Handshake method를 사용하여 Data 전송을 확인함

- Handshake : Agreement between two independent units
  - 1) Source-initiated handshake : ***Fig. 11-5***
  - 2) Destination-initiated handshake : ***Fig. 11-6***

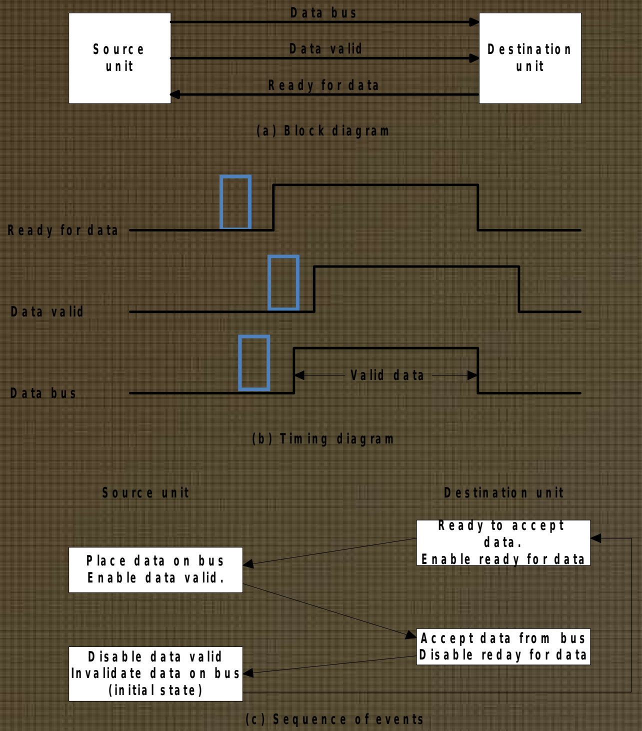


(a) Block diagram

(b) Timing diagram

(c) Sequence of events

Fig. 11-5 Source-initiated handshake



(a) Block diagram

(b) Timing diagram

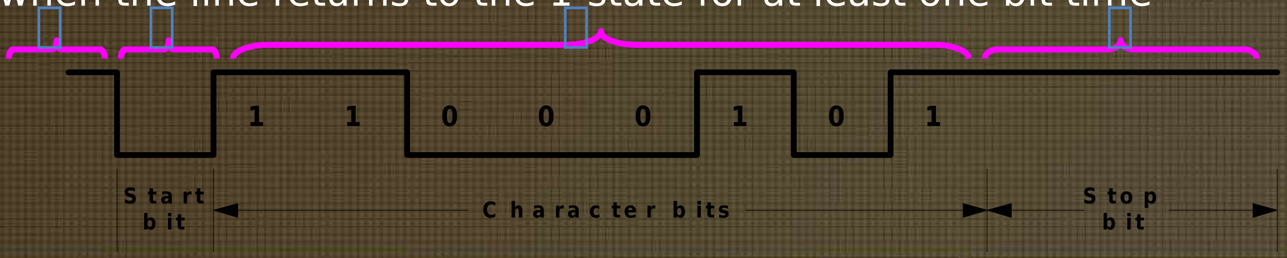
(c) Sequence of events

Fig. 11-6 Destination-initiated handsha

- **Timeout** : If the return handshake signal does not respond within a given time period, the unit assumes that an error has occurred.

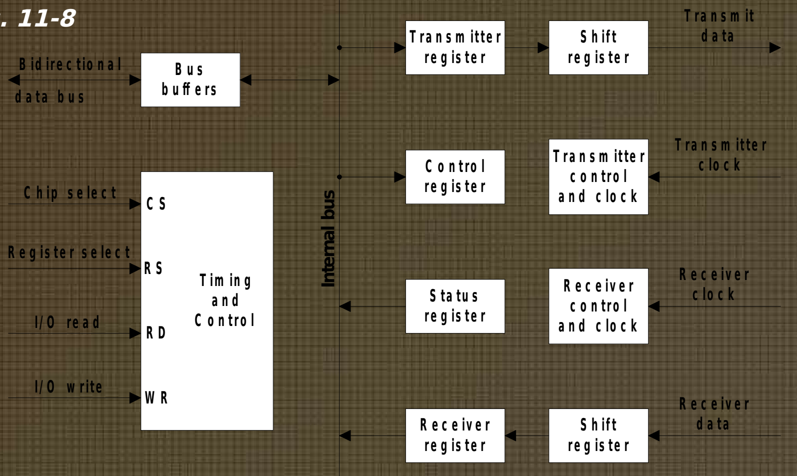
– Asynchronous Serial Transfer

- Synchronous transmission : ***Sec. 11-8***
  - – The two unit share a common clock frequency
  - – Bits are transmitted continuously at the rate dictated by the clock pulses
- Asynchronous transmission : ***Fig. 11-7***
  - – Special bits are inserted at both ends of the character code
  - – Each character consists of three parts :
    - » 1) start bit : always “0”, indicate the beginning of a character
    - » 2) character bits : data
    - » 3) stop bit : always “1”
- Asynchronous transmission rules :
  - – ▯When a character is not being sent, the line is kept in the 1-state
  - – ▯ The initiation of a character transmission is detected from the start bit, which is always “0”
  - – ▯ The character bits always follow the start bit
  - – ▯ After the last bit of the character is transmitted, a stop bit is detected when the line returns to the 1-state for at least one bit time



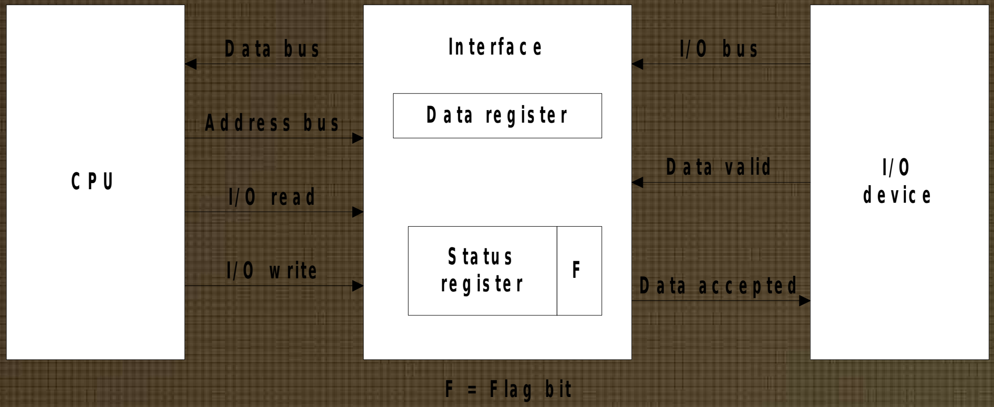
  - Baud Rate : Data transfer rate in bits per second
    - 10 character per second with 11 bit format = 110 bit per second
  - UART (Universal Asynchronous Receiver Transmitter) : 8250
  - UART (Universal Synchronous/Asynchronous Receiver Transmitter) : 8251

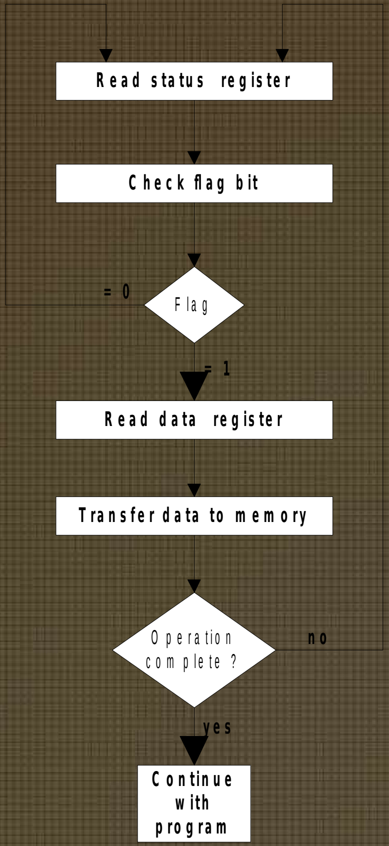
- Asynchronous Communication Interface : ***Fig. 11-8***
  - 8250 SIO
    - 80 : Data Write/Read (*Transmit/Receive*)
    - 81 : Control Write/ Status Read
      - » **A0** = RS (*register select*)
  - Double Buffered (*in transmit register*)
    - New character can be loaded as soon as the previous one starts transmission
  - 3 possible errors (*in status register*)
    - 1) parity error
      - » Even or Odd parity error
    - 2) framing error
      - » right number of stop bits is not detected at the end of the received character
    - 3) overrun error
      - » CPU does not read the character from the receiver register before the next one is available



CS	RS	Operation	Register selected
0	x	x	None : data bus in high-impedance
1	0	WR	Transmitter register
1	1	WR	Control register
1	0	RD	Receiver register
1	1	RD	Status register

- 11-4 Modes of Transfer
  - Data transfer to and from peripherals
    - 1) Programmed I/O : *in this section*
    - 2) Interrupt-initiated I/O : *in this section and sec. 11-5*
    - 3) Direct Memory Access (**DMA**) : *sec. 11-6*
    - 4) I/O Processor (**IOP**) : *sec. 11-7*
  - Example of Programmed I/O : ***Fig. 11-10, 11-11***



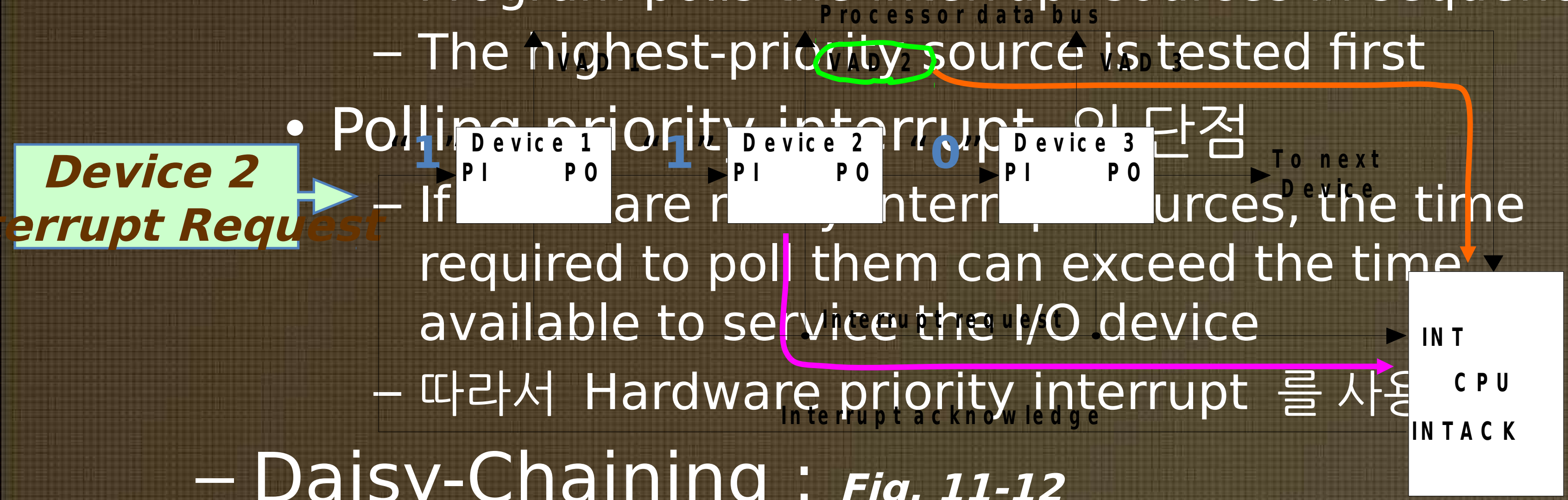


  - Interrupt-initiated I/O
    - 1) Non-vectored : fixed branch address
    - 2) Vectored : interrupt source supplies the branch address (**interrupt vector**)

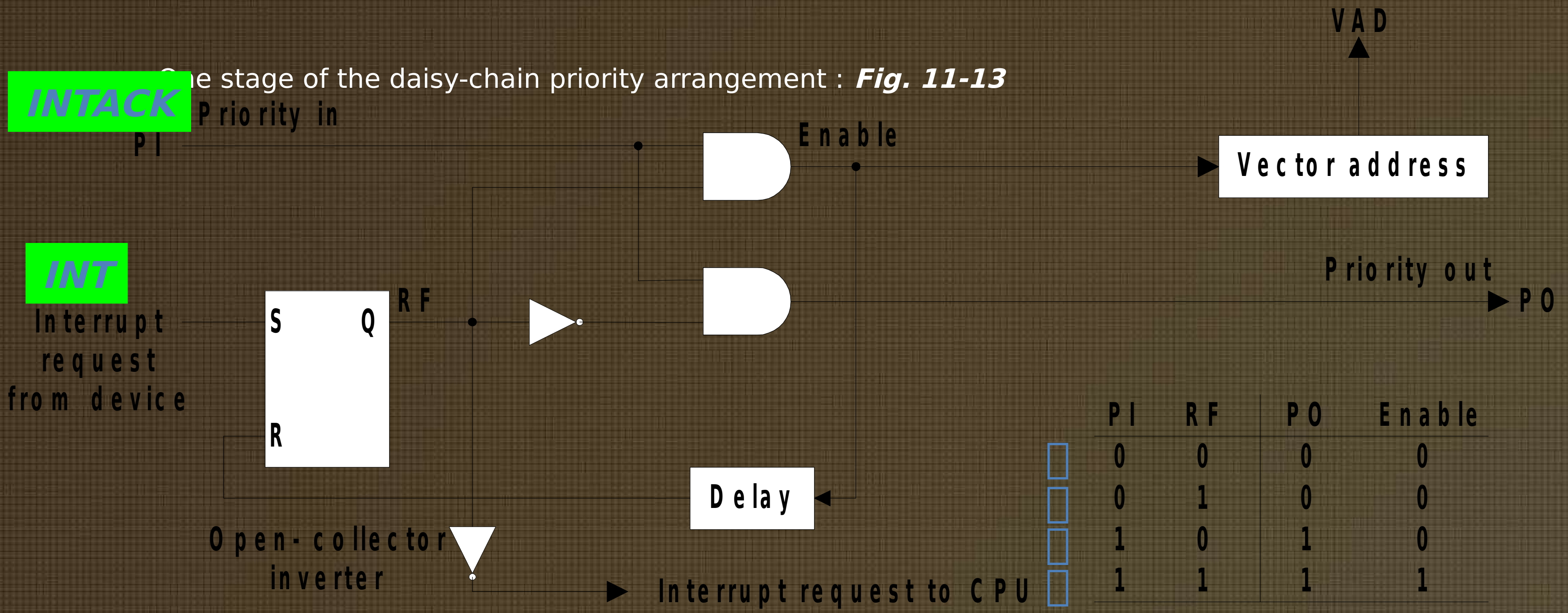
- Software Considerations
  - I/O routines
    - software routines for controlling peripherals and for transfer of data between the processor and peripherals
  - I/O routines for standard peripherals are provided by the manufacturer (**Device driver, OS** or **BIOS**)
  - I/O routines are usually included within the operating system
  - I/O routines are usually available as operating system procedures (**OS** or **BIOS function call**)

- 11-5 Priority Interrupt
  - Priority Interrupt
    - Identify the source of the interrupt when several sources will request service simultaneously
    - Determine which condition is to be serviced first when two or more requests arrive simultaneously
      - 1) Software : **Polling**
      - 2) Hardware : **Daisy chain**, **Parallel priority**

- Polling
  - Identify the highest-priority source by software means
    - One common branch address is used for all interrupts
    - Program polls the interrupt sources in sequence
    - The highest-priority source is tested first
  - Polling priority interrupt 의 단점
    - If there are many interrupt sources, the time required to poll them can exceed the time available to service the I/O device
    - 따라서 Hardware priority interrupt 를 사용
- Daisy-Chaining : ***Fig. 11-12***



One stage of the daisy-chain priority arrangement : ***Fig. 11-13***



	PI	RF	PO	Enable
□	0	0	0	0
□	0	1	0	0
□	1	0	1	0
□	1	1	1	1

□ No interrupt request

□ Invalid : interrupt request, but no acknowledge

□ No interrupt request : Pass to other device (*other device requested interrupt*)

□ Interrupt request

- Parallel Priority
  - Priority Encoder Parallel Priority : ***Fig. 11-14***
    - Interrupt Enable F/F (**IEN**) : set or cleared by the program
    - Interrupt Status F/F (**IST**) : set or cleared by the encoder output
  - Priority Encoder Truth Table : ***Tab. 11-2***
- Interrupt Cycle
  - At the end of each instruction cycle, CPU checks IEN and IST
  - if both IEN and IST equal to “1”
  - CPU goes to an Instruction le
    - Sequence of microoperation during Instruction Cycle

ranch to ISR

$SP \leftarrow SP - 1$: Decrement stack point

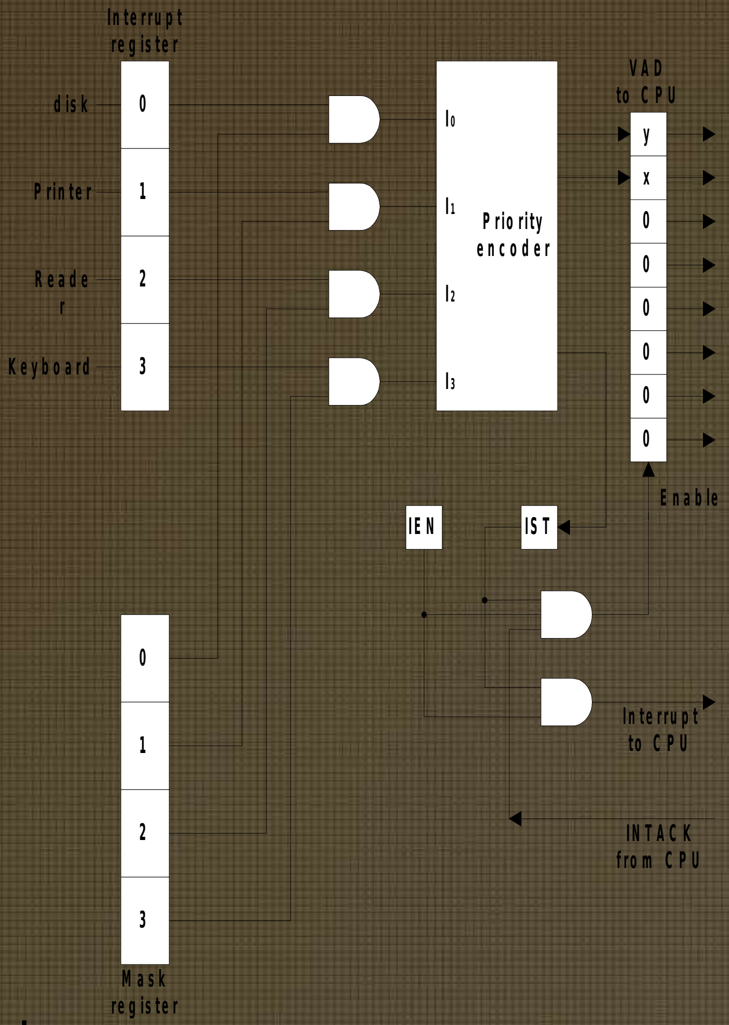
$M[SP] \leftarrow PC$: Push PC into stack

$INTACK \leftarrow 1$: Enable INTACK

$PC \leftarrow VAD$: Transfer VAD to PC

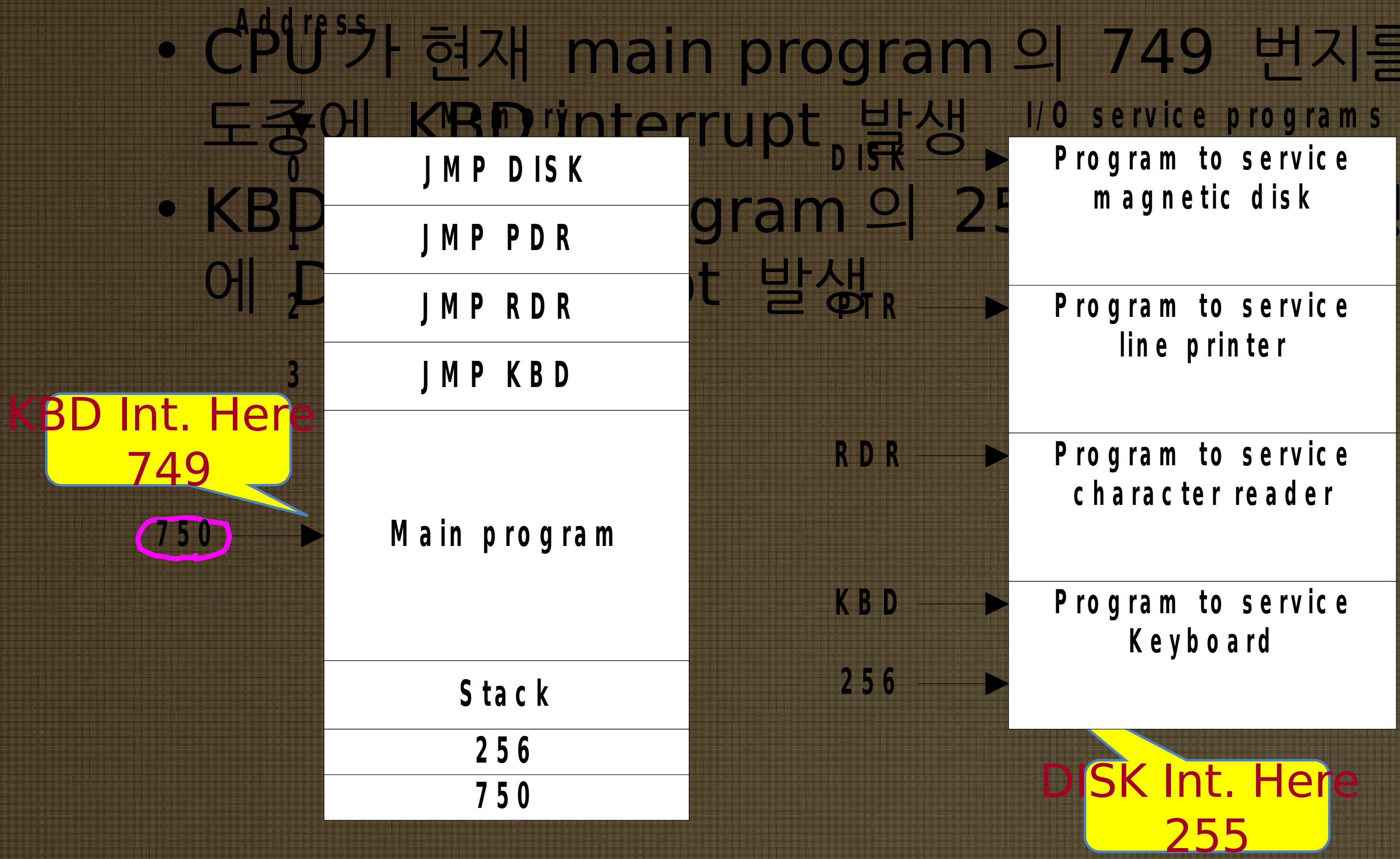
$IEN \leftarrow 0$: Disable further interrupts

Go to Fetch next instruction



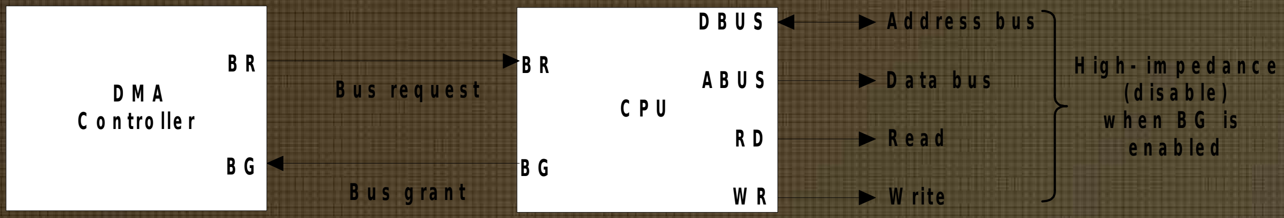
– Software Routines

- CPU 가 현재 main program 의 749 번지를 실행 도중에 KBD interrupt 발생
- KBD gram 의 25 도중 에 D t 발생

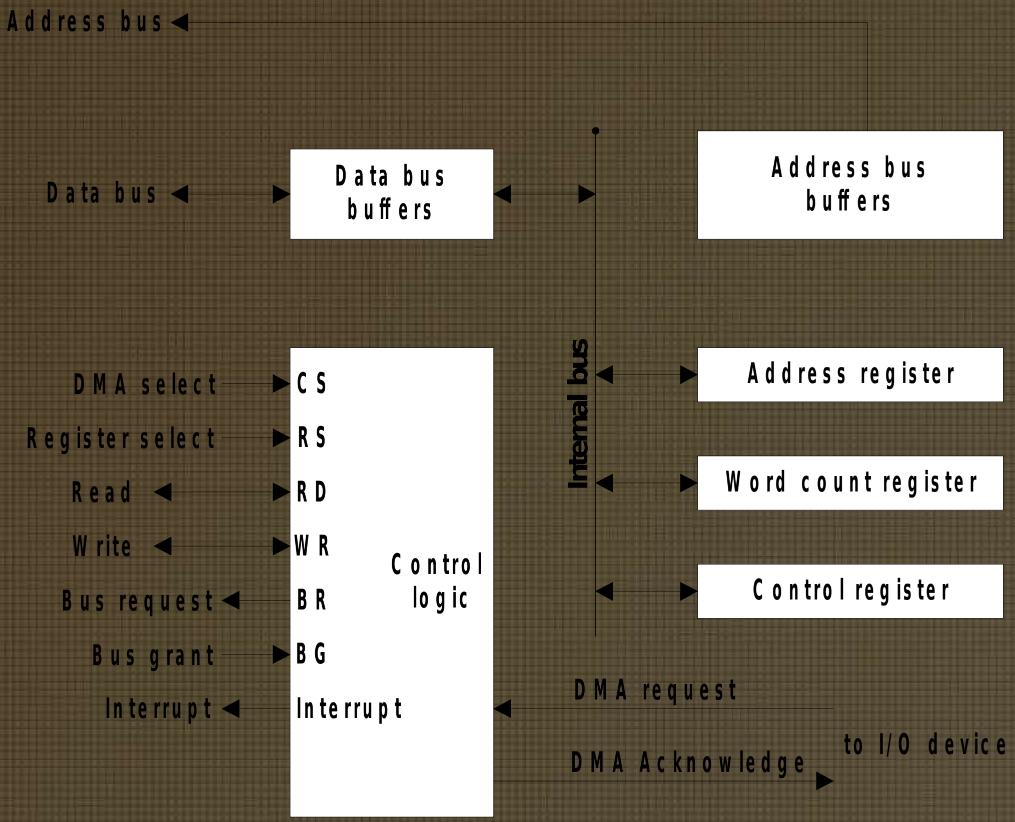


- 11-6 Direct Memory Access (DMA)
  - DMA
    - DMA controller takes over the buses to manage the transfer directly between the **I/O device** and **memory (Bus Request/Grant***)*

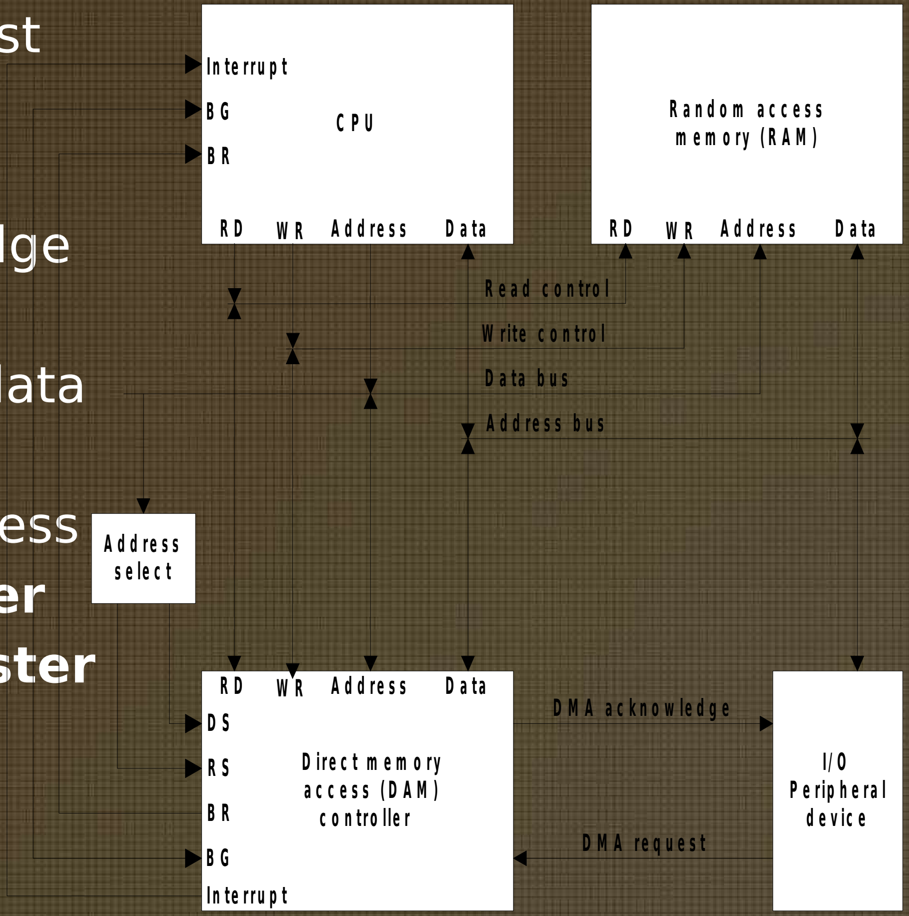
Fig. 11-14



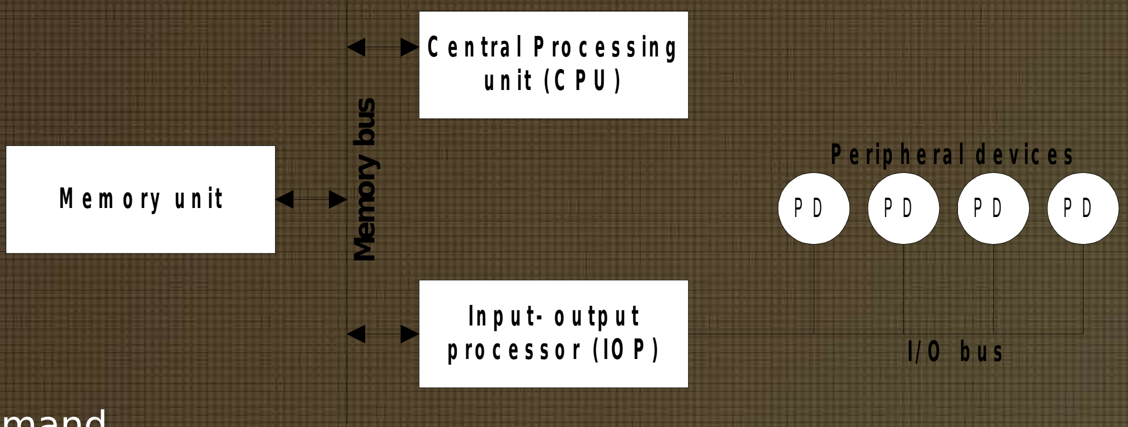
- Transfer Modes
  - 1) Burst transfer :
  - 2) Cycle stealing transfer :
- DMA Controller (Intel 8237 DMAC) : ***Fig. 11-17***
  - DMA Initialization Process
    - **1)** Set Address register :
      - » memory address for read/write
    - **2)** Set Word count register :
      - » the number of words to transfer
    - **3)** Set transfer mode :
      - » read/write,
      - » burst/cycle stealing,
      - » I/O to I/O,
      - » I/O to Memory,
      - » Memory to Memory
      - » Memory search
      - » I/O search
    - 4) DMA transfer start : *next section*
    - 5) EOT (End of Transfer) :
      - » Interrupt



- DMA Transfer (I/O to Memory)
  - 1) I/O Device sends a DMA request
  - 2) DMAC activates the **BR** line
  - 3) CPU responds with **BG** line
  - 4) DMAC sends a DMA acknowledge to the I/O device
  - 5) I/O device puts a word in the data bus (*for memory write*)
  - 6) DMAC write a data to the address specified by **Address register**
  - 7) Decrement **Word count register**
  - 8) **Word count register** **EOT** interrupt CPU
  - 9) **Word count register**
  - DMAC checks the DMA request from I/O device

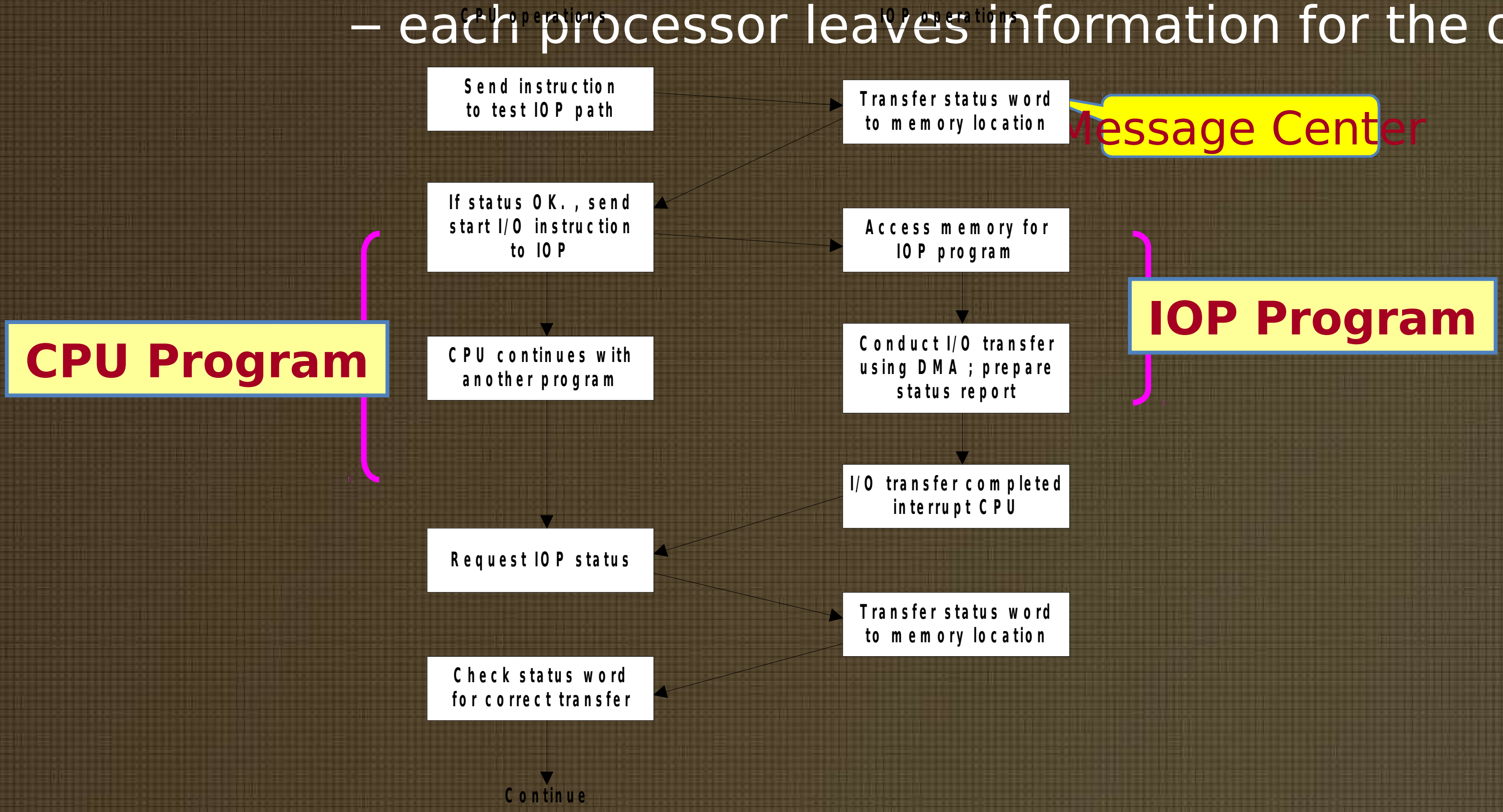


- 11-7 Input-Output Processor (IOP)
  - IOP : ***Fig. 11-19***
    - Communicate directly with all I/O devices
    - Fetch and execute its own instruction
      - IOP instructions are specifically designed to facilitate I/O transfer
      - DMAC must be set up entirely by the CPU
    - Designed to handle the details of I/O processing



  - Command
    - **Instruction** *that are read form memory by an* ***IOP***
      - Distinguish from instructions that are read by the CPU
      - Commands are prepared by experienced programmers and are stored in memory
      - Command word = IOP program

- CPU - IOP Communication : ***Fig. 11-20***
  - Memory units acts as a message center : Information
    - each processor leaves information for the other



CPU operations

IOP operations

Send instruction to test IOP path

Transfer status word to memory location

Message Center

If status O.K., send start I/O instruction to IOP

Access memory for IOP program

CPU Program

CPU continues with another program

Conduct I/O transfer using DMA; prepare status report

IOP Program

I/O transfer completed interrupt CPU

Request IOP status

Transfer status word to memory location

Check status word for correct transfer

Continue

- 11-8 Serial Communication
  - Difference between I/O Processor and Data Communication Processor
    - I/O Processor
      - communicate with peripherals through a common I/O bus (data, address, control bus)
    - Data Communication Processor
      - communicate with each terminal through a single pair of wires
  - Modem (= *Data Sets, Acoustic Couplers*)
    - Convert digital signals into audio tones to be transmitted over telephone lines
    - Various modulation schemes are used (FM, AM, PCM)
  - Block transfer
    - An entire block of characters is transmitted in synchronous transmission
    - Transmitter sends **one more character** (**error check**) after the entire block is sent
  - Error Check
    - LRC (Longitudinal Redundancy Check) : XOR
    - CRC (Cyclic Redundancy Check) : Polynomial
  - 3 Transmission System
    - Simplex : one direction only
    - Half-duplex : both directions but only one direction at a time
    - Full-duplex : both directions simultaneously

- Data Link
  - The communication lines, modems, and other equipment used in the transmission of information between two or more stations
- Data Link Protocol
  - 1) Character-Oriented Protocol
  - 2) Bit-Oriented Protocol
- Character-Oriented Protocol
  - Message format for Character-Oriented Protocol : ***Fig. 11-25***

SYN	**SYN**	**SOH**	**Header**	**STX**	**Text**	**ETX**	**BCC**

    - TEXT :
    - BCC : Block Check Character (LRC or CRC)
  - ASCII Communication Control Character : ***Tab. 11-4***
    - **SYN** (0010110) : Establishes synchronism
    - **SOH** (0000001) : Start of Header (address or control information)
    - **STX** (0000010) : Start of Text
    - **ETX** (0000011) : End of Text
  - Transmission Example : ***Tab. 11-5, 11-6***

Assignment 4

Q1. Differentiate synchronous and asynchronous transmission.

Q2. What is CAM?

Q3. Give the block diagram of DMA controller. Why are the read and write control lines in a DMA controller bidirectional?

Q4. Explain the working principle of I/O processors.

Q5. Discuss the Programmed I/O method for controlling input output operations.

Tutorial 4

Q1. Differentiate synchronous and asynchronous communication.

Q2. What are various asynchronous communication protocols explain.

Q3. What is interrupt? Explain priority interrupt.

Q4. What are various mode of data transfer?

Q5. What do you mean by I/O processors?

Outcomes

After reading above topics students will be able to:

- Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

COMPUTER ORGANIZATION & ARCHITECTURE

UNIT 5

Pipelining & Speed Up Concept

Presentation Outline

- Architectual Classification
  - Flynn’s
  - Feng’s
- Pipelining
  - Concepts
  - Performance
- Speed Laws
- Pipelining Hazards

Learning Objectives

The objectives of the following slide is to make student aware about the :

- Architectural Classification
  - Flynn’s
  - Feng’s
- Pipelining
  - Concepts
  - Performance

Characterize Pipelines

1) Hardware or software implementation – pipelining can be implemented in either software or hardware.
2) Large or Small Scale – Stations in a pipeline can range from simplistic to powerful, and a pipeline can range in length from short to long.
3) Synchronous or asynchronous flow – A synchronous pipeline operates like an assembly line: at a given time, each station is processing some amount of information. A asynchronous pipeline, allow a station to forward information at any time.
4) Buffered or unbuffered flow – One stage of pipeline sends data directly to another one or a buffer is place between each pairs of stages.
5) Finite Chunks or Continuous Bit Streams – The digital information that passes though a pipeline can consist of a sequence or small data items or an arbitrarily long bit stream.
6) Automatic Data Feed Or Manual Data Feed – Some implementations of pipelines use a separate mechanism to move information, and other implementations require each stage to participate in moving information.

What is Pipelining

- A technique used in advanced microprocessors where the microprocessor begins executing a second instruction before the first has been completed.

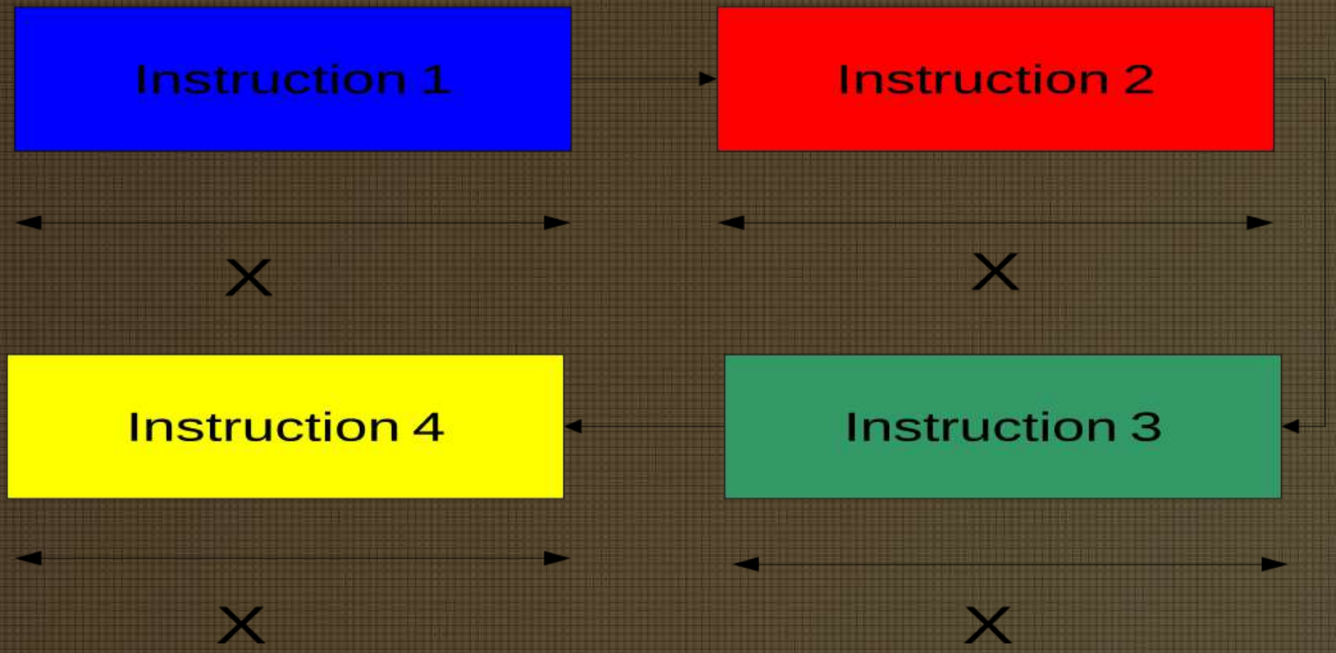
- A Pipeline is a series of stages, where some work is done at each stage. The work is not finished until it has passed through all stages.

- With pipelining, the computer architecture allows the next instructions to be fetched while the processor is performing arithmetic operations, holding them in a buffer close to the processor until each instruction operation can performed.

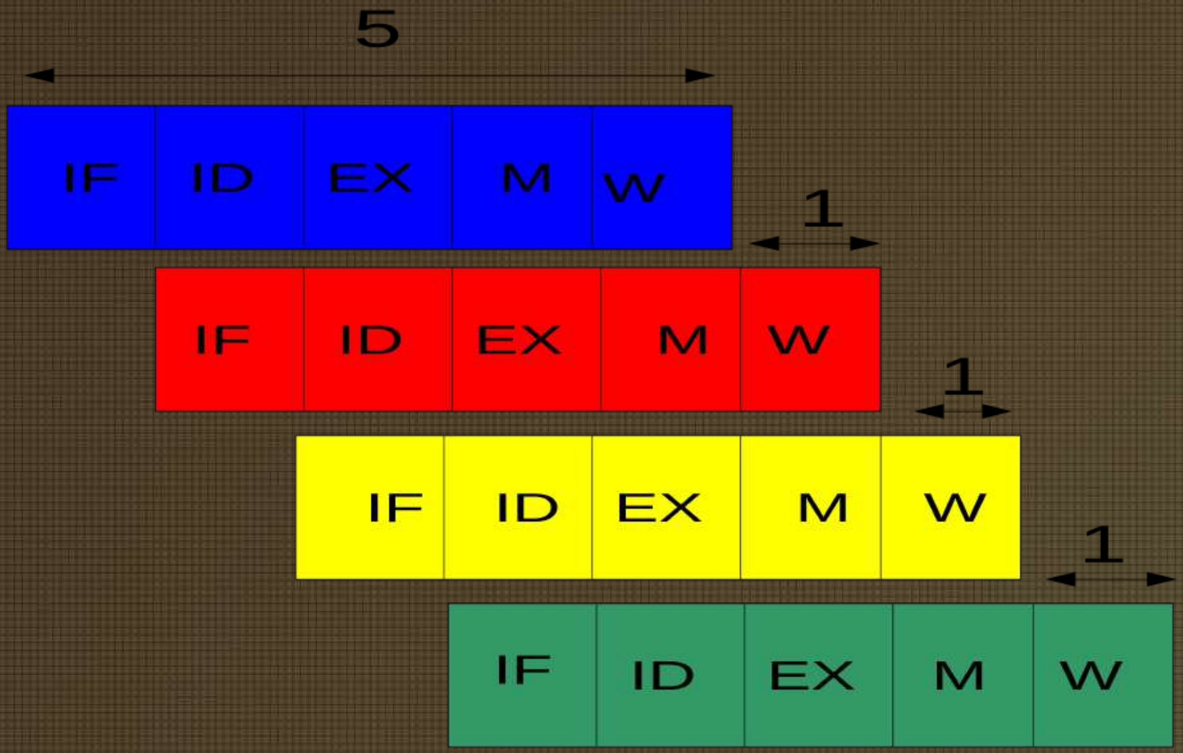
How Pipelines Works

- The pipeline is divided into segments and each segment can execute it operation concurrently with the other segments. Once a segment completes an operations, it passes the result to the next segment in the pipeline and fetches the next operations from the preceding segment.

Example



Four sample instructions, executed linearly



5

IF ID EX M W

1

IF ID EX M W

1

IF ID EX M W

1

IF ID EX M W

Four Pipelined Instructions

Instructions Fetch

- The instruction Fetch (IF) stage is responsible for obtaining the requested instruction from memory. The instruction and the program counter (which is incremented to the next instruction) are stored in the IF/ID pipeline register as temporary storage so that may be used in the next stage at the start of the next clock cycle.

Instruction Decode

- The Instruction Decode (ID) stage is responsible for decoding the instruction and sending out the various control lines to the other parts of the processor. The instruction is sent to the control unit where it is decoded and the registers are fetched from the register file.

Execution

- The Execution (EX) stage is where any calculations are performed. The main component in this stage is the ALU. The ALU is made up of arithmetic, logic and capabilities.

Memory and IO

- The Memory and IO (MEM) stage is responsible for storing and loading values to and from memory. It also responsible for input or output from the processor. If the current instruction is not of Memory or IO type than the result from the ALU is passed through to the write back stage.

Write Back

- The Write Back (WB) stage is responsible for writing the result of a calculation, memory access or input into the register file.

Operation Timings

- Estimated timings for each of the stages:

Instruction Fetch	2ns
Instruction Decode	1ns
Execution	2ns
Memory and IO	2ns
Write Back	1ns

Advantages/Disadvantages

Advantages:

- More efficient use of processor
- Quicker time of execution of large number of instructions

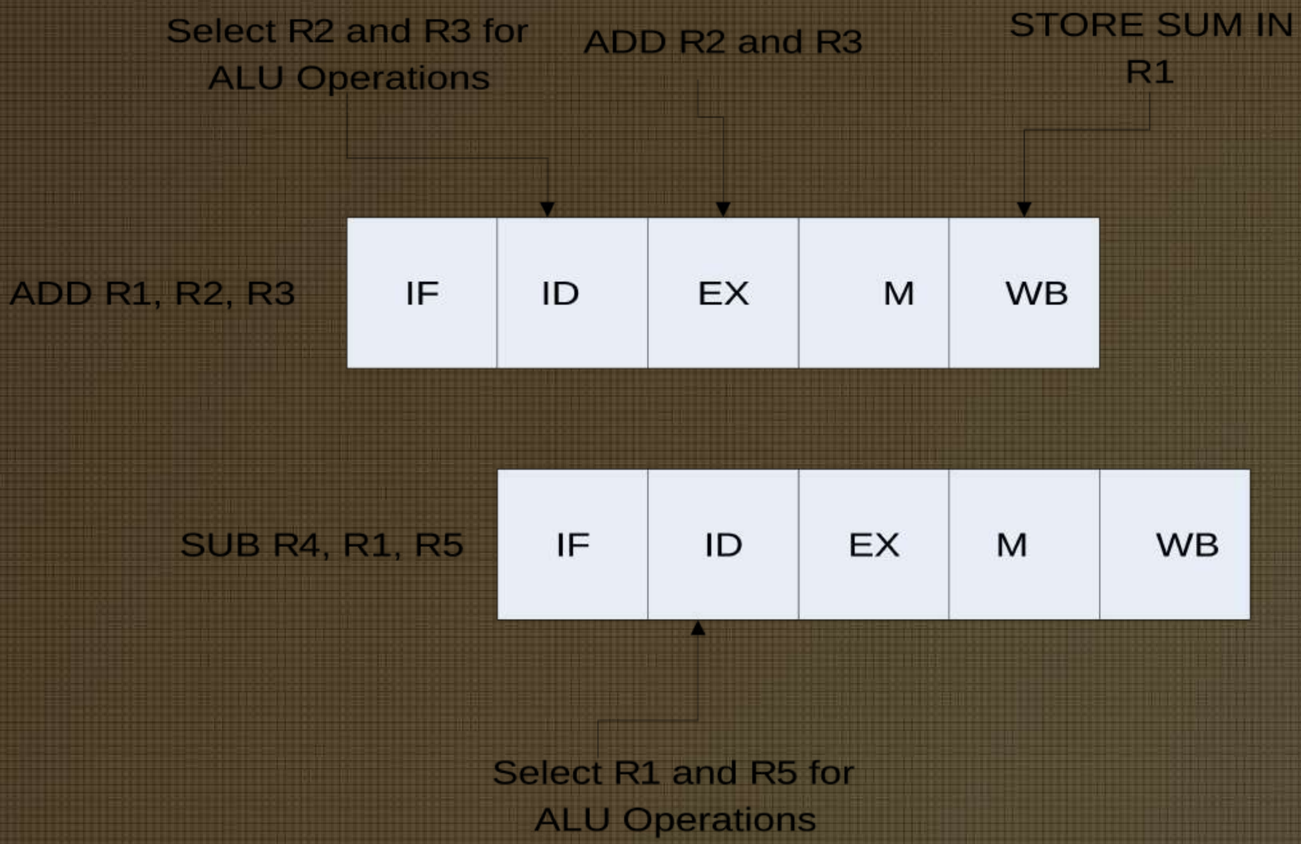
Disadvantages:

- Pipelining involves adding hardware to the chip
- Inability to continuously run the pipeline at full speed because of pipeline hazards which disrupt the smooth execution of

Pipeline Hazards

- Data Hazards – an instruction uses the result of the previous instruction. A hazard occurs exactly when an instruction tries to read a register in its ID stage that an earlier instruction intends to write in its WB stage.
- Control Hazards – the location of an instruction depends on previous instruction
- Structural Hazards – two instructions need to access the same resource

Data Hazards



Select R2 and R3 for ALU Operations

ADD R2 and R3

STORE SUM IN R1

ADD R1, R2, R3

IF ID EX M WB

SUB R4, R1, R5

IF ID EX M WB

Select R1 and R5 for ALU Operations

Stalling

- Stalling involves halting the flow of instructions until the required result is ready to be used. However stalling wastes processor time by doing nothing while waiting for the result.

ADD R1, R2, R3	IF	ID	EX	M	WB				
STALL		IF	ID	EX	M	WB			
STALL			IF	ID	EX	M	WB		
STALL				IF	ID	EX	M	WB	
SUB R4, R1, R5					IF	ID	EX	M	WB

Type of Pipelining

- Software Pipelining
  1) Can Handle Complex Instructions
  2) Allows programs to be reused
- Hardware Pipelining
  1) Help designer manage complexity – a complex task can be divided into smaller, more manageable pieces.
  2) Hardware pipelining offers higher performance

Type of Hardware Pipelines

- Instruction Pipeline - An **instruction pipeline** is very similar to a manufacturing assembly line.

1st stage receives some parts, performs its assembly task, and passes the results to the second stage;

2nd stage takes the partially assembled product from the first stage, performs its task, and passes its work to the third stage;

3rd stage does its work, passing the results to the last stage, which completes the task and outputs its results.

- Data Pipeline – data pipeline is designed to pass data from stage to stage.

Instruction Pipelines Conflict

- It divided into two categories.
  - Data Conflicts
  - Branch Conflicts

- When the current instruction changes a register that the next one needed, data conflicts happens.

- When the current instruction make a jump, branch conflicts happens.

ARCHITECTURAL CLASSIFICATION

- **Flynn classification**: (1966) is based on multiplicity of instruction streams and the data streams in computer systems.
- **Feng’s classification**: (1972) is based on serial versus parallel processing.

Flynn classification:

- It Is based on multiplicity of instruction streams and the data streams in computer systems.
- The most popular taxonomy of computer architecture was defined by Flynn.
  - Flynn's classification scheme is based on the notion of a stream of information. Two types of information flow into a processor: instructions and data.
  - The instruction stream is defined as the sequence of instructions performed by the processing unit.
  - The data stream is defined as the data traffic exchanged between the memory and the processing unit
- Computer architecture can be classified into the following four distinct categories:
  - single-instruction single-data streams (**SISD**);
  - single-instruction multiple-data streams (**SIMD**);
  - multiple-instruction single-data streams (**MISD**); and
  - multiple-instruction multiple-data streams (**MIMD**).

FENG’S CLASSIFICATION

- Tse-yun Feng suggested the use of degree of parallelism to classify various computer architectures.
- The maximum number of binary digits that can be processed within a unit time by a computer system is called the maximum parallelism degree P.
- A bit slice is a string of bits one from each of the words at the same vertical position.
- Under above classification
  - Word Serial and Bit Serial (WSBS)
  - Word Parallel and Bit Serial (WPBS)
  - Word Serial and Bit Parallel(WSBP)
  - Word Parallel and Bit Parallel (WPBP)
- WSBS has been called bit parallel processing because one bit is processed at a time.
- WPBS has been called bit slice processing because m-bit slice is processes at a time.
- WSBP is found in most existing computers and has been called as Word Slice processing because one word of n bit processed at a time.
- WPBP is known as fully parallel processing in which an array on n x m bits is processes at one time.

Assignment 5

Q1. Explain pipeline concept with performance matrices.

Q2. What do you mean by delayed branch?

Q3. Explain Flynn’s and Feng’s classification.

Q4. What is Amdahl's Law & Gustafson's Law for speed up.

Q5. What are the various hazard in pipelining? How they can be resolved?

Tutorial 5

Q1. Differentiate linear and non linear pipeline processors.

Q2. What are various pipeline conflicts?

Q3. A non-pipeline system takes 50 ns to process a task. The same task can processed in a six-segment pipeline with a dock cycle of 10 ns. Determine the speedup ratio of the pipeline for 100 tasks. What is the maximum speedup that can achieved?

Q4.Give an example of program that will cause data conflict in the three segment pipeline.

Q5. Discuss various issues in instruction pipelining.

Outcomes

After reading above topics students will be able to:

- Ability to Identify, formulates, review research literature and analyze complex engineering problems reaching substantiated conclusions using first principles mathematics, natural science and engineering science.
- Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.