SECR2033
Computer Organization and Architecture

Module 5
Central Processing Unit (CPU)

Objectives:

- To learn the components common to every CPU.
- Be able to explain how each component in CPU contributes to instruction cycle.
- To understand the concept of pipelining in CPU execution.
- Be able to understand micro-operations basis for the design and implementation of the control unit.

Module 5

Central Processing Unit (CPU)

3

5.1 Processor Organization
5.2 Register Organization
5.3 Instruction Cycles
5.4 Instruction Pipelining
5.5 Control Unit Operation
5.6 Microprogrammed Control
5.7 Summary

Module 5a
Central Processing Unit (CPU)

5

5.1 Processor Organization
- Overview

5.2 Register Organization

5.3 Instruction Cycles

5.4 Instruction Pipelining

5.5 Control Unit Operation

5.6 Microprogrammed Control

5.7 Summary

5 Overview

- To understand the organization of the processor, let us consider the requirements placed on the processor, the things that it must do:
  - *Fetch instruction*: reads an instruction from memory (register, cache, main memory).
  - *Interpret instruction*: The instruction is _____ to determine what action is required.
  - *Fetch data*: The execution of an instruction may require reading data from memory or an I/O module.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.489. 6

5

- *Process data*: The execution of an instruction may require performing some arithmetic or logical operation on data.
- *Write data*: The results of an execution may require writing data to memory or an I/O module.

- To do these things, the processor needs a small _____ to store some data and instruction temporarily while an instruction is being executed.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.489. 7

5

Minimal internal memory, consisting of a set of storage locations, called registers.

ALU does the actual computation or processing of data

Control unit controls the movement of data and instructions into / out of the processor and controls the operation of the ALU

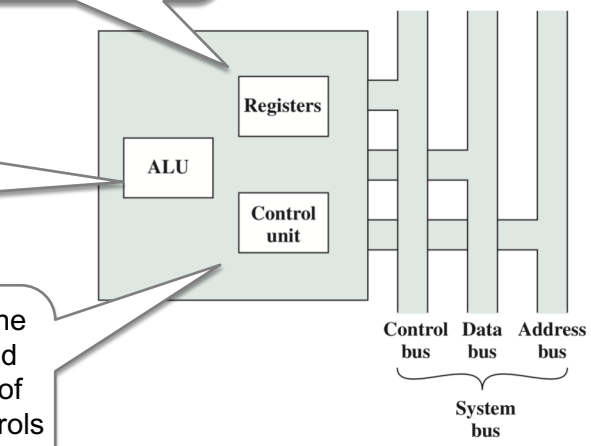


Figure: The CPU with the System Bus.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.490. 8

- The computer's CPU *fetches*, *decodes*, and *executes* program instructions.
- The two principal parts of the CPU:

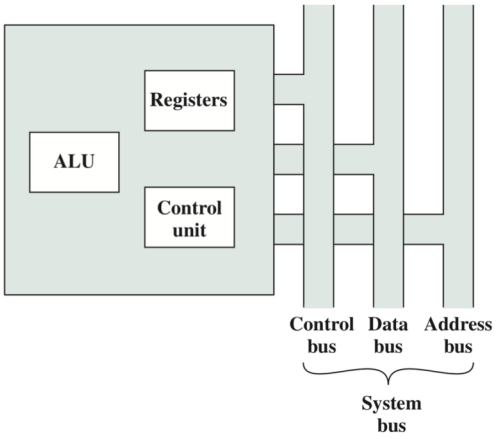


Figure: The CPU with the System Bus.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.490. 9

1) The **data path** consists of an *ALU* and storage units (*registers*) that are interconnected by a *data bus* that is also connected to *main memory*.
2) The **control unit** provides signals to tell the **data path**, *memory*, and *I/O* devices what to do according the instructions of the program.

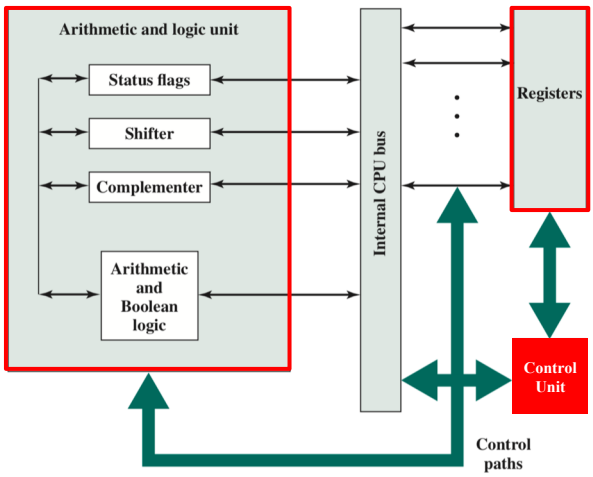


Figure: Internal Structure of the CPU.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.490. 10

Internal CPU Bus

- The *registers* are interconnected, and connected with *main memory* through a common data bus.
- Each device on the bus is identified by a unique number that is set on the control lines whenever that device is required to carry out an operation.

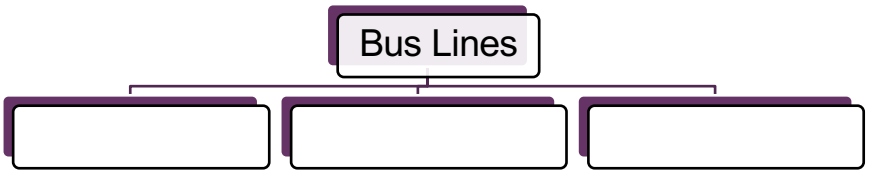
This permits data transfer between these devices without use of the main data bus.

- Separate connections are also provided between:
  - the accumulator & the memory buffer register, and
  - the ALU & the accumulator & memory buffer register.

11

External Bus

- A bus is a set of wires that simultaneously convey a single bit along each line.



12

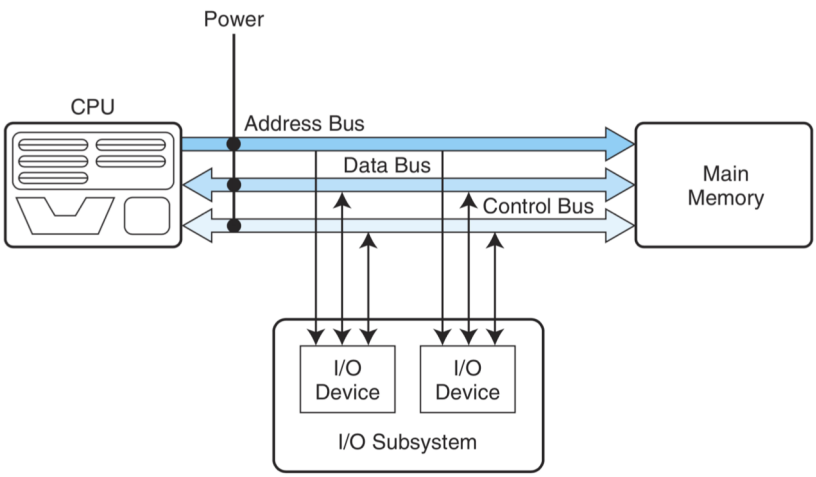


Figure: The Components of a Typical Bus

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.149 13

1) **Data lines**
   - convey bits from one device to another.
   - The number of lines (32, 64,128) gives the width of the data bus.
   - **Example**: width = 32 bit, instruction length = 64 bits → this means the processor must access the memory modules twice during each instruction cycle.

2) **Control lines**
   - determine the direction of data flow, and when each device can access the bus.
   - Uses control signals, timing signals, command signals → memory write, memory read, bus request and bus grant.

14

3) Address lines

- determine the location of the source or destination of the data.
- **Example**: want to read data in memory, put address on this bus, go to memory, get the content (*i.e.* the data) and put it on to data bus.

15

Module 5a
Central Processing Unit (CPU)

17

5.1 Processor Organization

5.2 Register Organization

- Overview
- User-Visible Registers
- Control & Status Registers

5.3 Instruction Cycles

5.4 Instruction Pipelining

5.5 Control Unit Operation

5.6 Microprogrammed Control

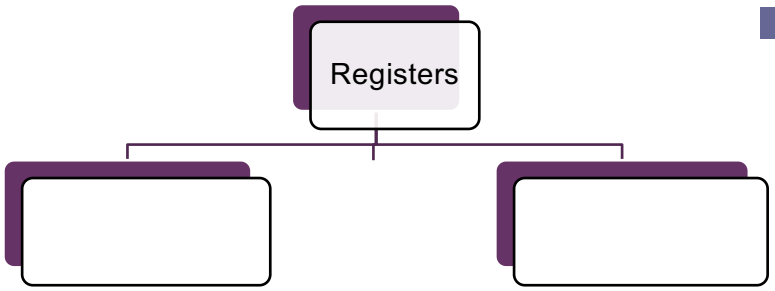
5.7 Summary

5 Overview

- Registers form the highest level of the _____.
  - Small set of high speed storage locations.
  - Temporary storage for data and control information.
- Registers hold data that can be readily accessed by the CPU.
- They can be implemented using D flip-flops.
  - A 32-bit register requires 32 D flip-flops.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.491. 18

5



(*General Purpose*)

May be referenced by assembly-level instructions and are thus "visible" to the user.

(*Special Purpose*)

Used to control the operation of the CPU. Most are not visible to the user.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.491. 19

5

(a) User-Visible Registers

- This register will be referenced by means of the machine language that the processor executes.
- Categories:
  - General purpose
  - Data
  - Address
  - Condition codes

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.491. 20

5

General purpose registers:

- Can be assigned to a variety of functions.
- Defined to the operations within the instructions.
- Can be used for addressing functions.
- **Examples**: accumulator, base, count, data.

Data registers:

- Hold data and cannot be used in the calculation of an operand address.
- **Example**: accumulator.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.491-492. 21

5

Address registers:

- Hold address information.
- **Examples**: general purpose address registers, segment pointers, stack pointers, index registers.

Condition codes or _____ :

- Bits set by the processor hardware as a result of operations.
- Can be accessed by a program but not changed directly.
- **Examples**: *Sign Flag* (SF), *Zero Flag* (ZF), *Overflow Flag* (OF).
- Bit values are used as the basis for conditional jump instructions.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.492. 22

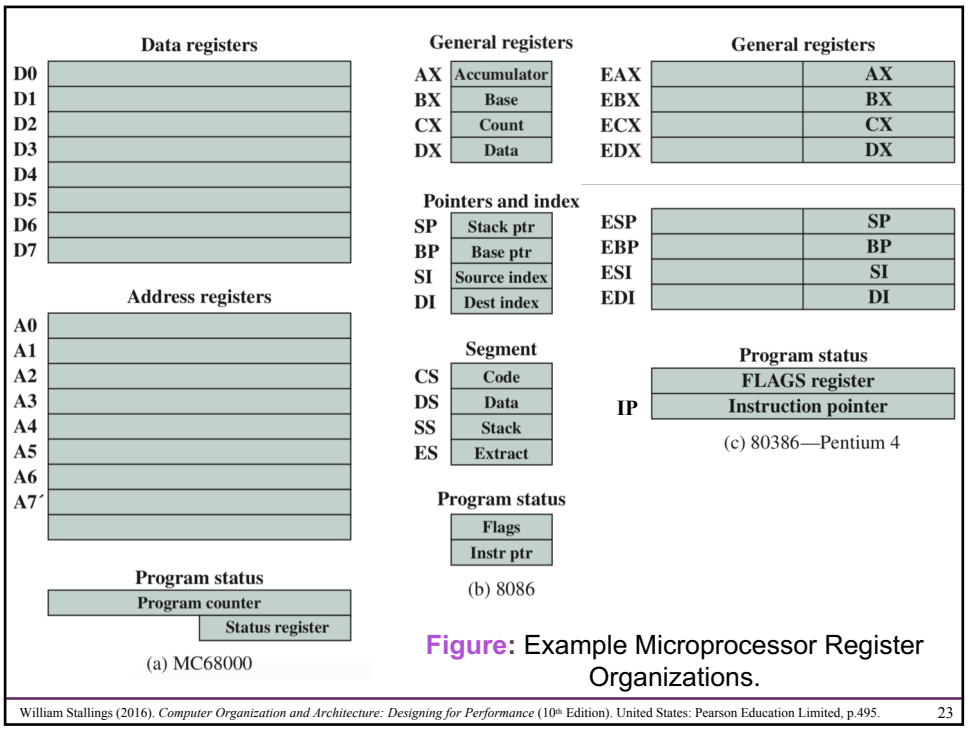
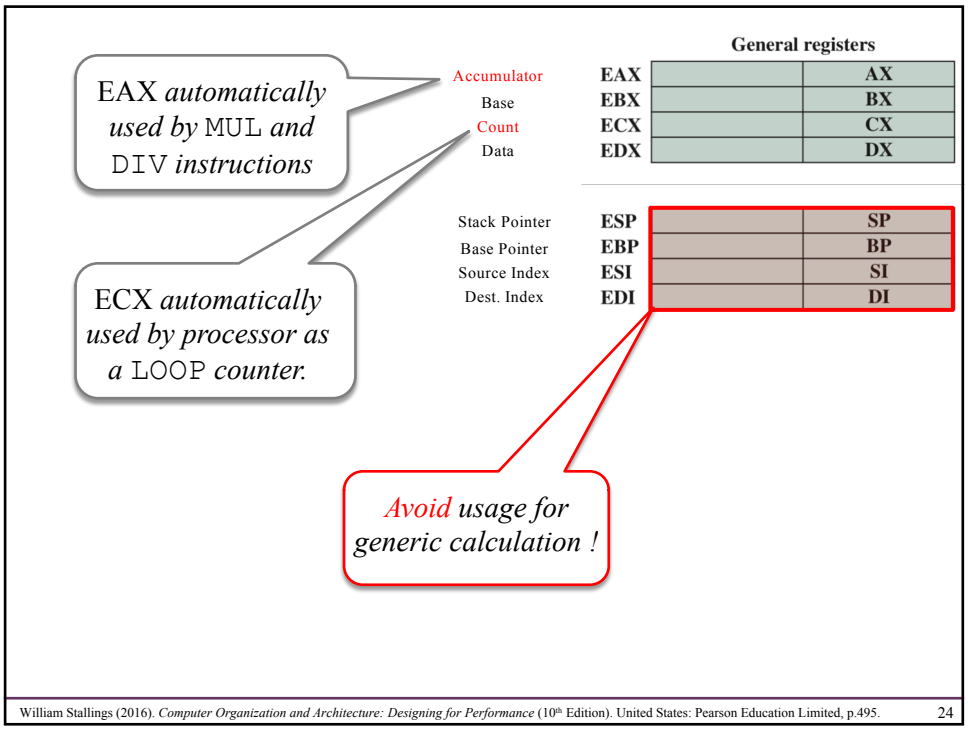


Figure: Example Microprocessor Register Organizations.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.495. 23



William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.495. 24

5

- Address registers should be wide enough to hold the longest address!
- Data registers should be wide enough to hold most data types.
- Would not want to use 64-bit registers if the vast majority of data operations used 16 and 32-bit operands.
- Related to width of memory data bus.

- **Solution**: _____ registers together to store longer formats (DX : AX)

25

Example:

```
mul   ax
```

Assuming the following *DumpRegs*. Update the related register once the instruction executed:

```
EAX=770C0400  EBX=7FFD7098  ECX=00000000  EDX=00401005
ESI=00000000  EDI=00000000  EBP=0012FF94  ESP=0012FF8C
EIP=00401025  EFL=00000A82  CF=0  SF=1  ZF=0  OF=1
```

27

Solutions:

```
mul   ax
```

→ ax = _____ h

→ ax * ax = 0400 * 0400 = 0010 0000h
DX : AX

```
EAX=770C0000  EBX=7FFD7098  ECX=00000000  EDX=00400010
ESI=00000000  EDI=00000000  EBP=0012FF94  ESP=0012FF8C
EIP=00401025  EFL=00000A82  CF=0  SF=1  ZF=0  OF=1
```

28

Example: `div bx`

Assuming the following *DumpRegs*. Update the related register once the instruction executed:

```
EAX=770C0000  EBX=7FFD0020  ECX=00000000  EDX=00400010
ESI=00000000  EDI=00000000  EBP=0012FF94  ESP=0012FF8C
EIP=00401025  EFL=00000A82  CF=0  SF=1  ZF=0  OF=1
```

29

Solutions: `div bx`

→ dx : ax = _____ h, bx = 0020h

→ [dx : ax] / bx = 100000 / 0020 = 0000 8000h

DX : AX

```
EAX=770C8000  EBX=7FFD0020  ECX=00000000  EDX=00400000
ESI=00000000  EDI=00000000  EBP=0012FF94  ESP=0012FF8C
EIP=00401025  EFL=00000A82  CF=0  SF=1  ZF=0  OF=1
```

30

5

(b) Control & Status Registers

- There are a variety of processor registers that are employed to control the operation of the processor.
- Most of these, on most machines, are not visible to the user.
- Some of them may be visible to machine instructions executed in a control or operating system mode.

Different machines will have different register organizations and use different terminology.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.493. 31

CPU (Control Processing Unit)
I/O (Input/Output)

5

Table: Four essential registers for instruction execution.

Registers	Function
(PC)	The address of an instruction to be fetched.
Instruction Register (IR)	The instruction most recently fetched.
Memory Address Register (MAR)	The address of a location in memory.
Memory Buffer Register (MBR)	A word of data to be written to memory or the word most recently read.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.493. 32

5

- The four registers just mentioned are used for the movement of data between the _____ and _____.
- Not all processors have internal registers designated as MAR and MBR → need some equivalent buffering mechanism to store the bits transferred / read bits to / from the data bus:
  - The ALU may have direct access to the MBR and user-visible registers.
  - There may be additional buffering registers at the boundary to the ALU serves as input and output registers for the ALU and exchange data with the MBR and user-visible registers.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.494. 33

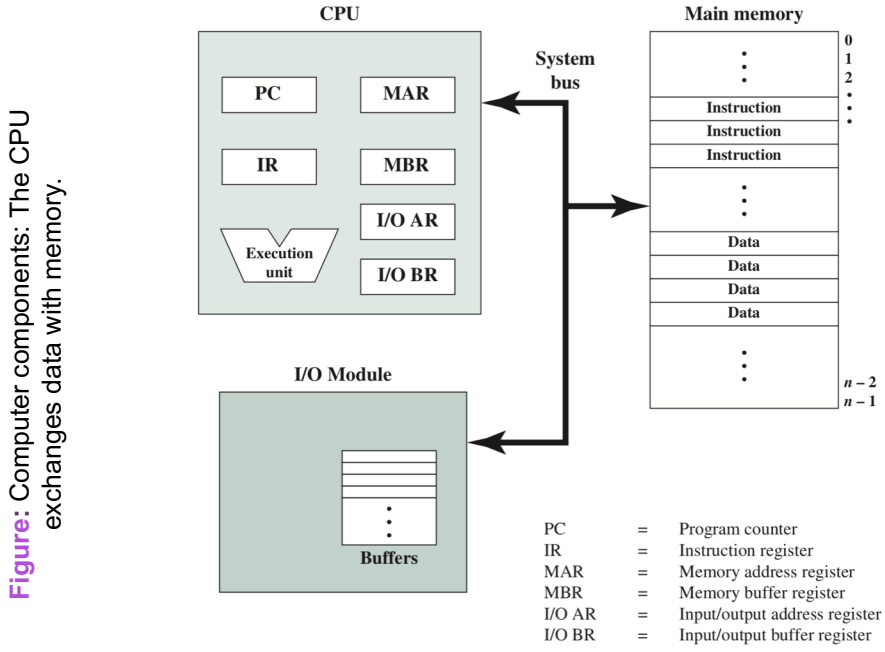


Figure: Computer components: The CPU exchanges data with memory.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.84. 34

Instruction Pointer

5

- The *Extended Instruction Pointer* (EIP) register holds the address of the next instruction to be executed.
- The EIP register corresponds to the _____ (PC) register in other architectures.
- EIP can be manipulated for certain instructions (*e.g.* call, `jmp`, `ret`) to branch to a new location.

35

Program Status

5

- Many processor designs include a register or set of registers, often known as the *Program Status Word* (PSW), that contain status information and condition codes.
- The ALU has a number of _____ that reflect the outcome of arithmetic (and bitwise) operations.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.494. 37

5

Status flags that reflect the outcome of arithmetic (contents of the destination operand)

- Common field or flags:
  - _____ (ZF): set when destination (result) equals zero.
  - _____ (SF): set when destination (result) is negative.
  - _____ (CF): set when unsigned value is out of range.
  - _____ (OF): set when signed value is out of range.
  - _____ (AF): set when there is carry from lower nibble to higher nibble in the lower byte. (bit 3 to bit 4)
  - _____ (PF): set when destination (result) has even number of 1's in the lower byte.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.494. 38

Example 1: Flags

Assuming that the system uses 8-bit system.

→ 06h + 54h

ZF : 0
SF : 0
CF : 0
OF : 0
AF : 0
PF :

```
(Carry)  0   0
     0000 0110  (06h)
+    0101 0100  (54h)
    ---------------
     0101 1010  (5Ah)
```

Operation result → not zero (0) ; so ZF = 0

Number of 1s = 4 → even ; so PF = _____

39

Example 2: Flags

Assuming that the system uses 8-bit system.
→ 2Fh + DEh

ZF : 0
SF : 0
CF :
OF : 0
AF : 1
PF : 0

```
      1
     0010 1111  (2Fh)
+    1101 1110  (DEh)
    ---------------
   1 0000 1101
```

There is a carry from lower nibble to higher nibble in the lower byte → so AF = 1

There is a carry out → so CF = ____

Number of 1s = 3 → odd ; so PF = 0

40

Example 3: Flags

a) Assuming that the system uses a word sized number for:

b) Assuming that the system uses a doubleword sized number for:

→ 5433h + 7098h

ZF :
SF :
CF :
OF :
AF :
PF :

ZF :
SF :
CF :
OF :
AF :
PF :

41

Solutions (a):

a) A word sized number.

→ 5433h + 7098h

ZF : ☐
SF : ☐
CF : ☐
OF : ☐
AF : ☐
PF : ☐

```
    0101 0100 0011 0011
+   0111 0000 1001 1000
  --------------------
  |                    |
```

42

Solutions (a):

a) Assuming that the system uses a word sized number for:

ZF :
SF :
CF :
OF :
AF :
PF :

→ 5433h + 7098h

```
(Carry)  0   1
         0101 0100 0011 0011
+        0111 0000 1001 1000
---------------------------
         
```

= -15157_{10}

Overflow occurred and lead to incorrect answer. Solution → The system must uses larger bit representation.

43

New

Solutions (a): new

a) Assuming that the system uses a 17-bit sized number for:

ZF :
SF :
CF :
OF :
AF :
PF :

→ 5433h + 7098h = $C4CB_{16} = 50379_{10}$

```
(Carry)  0   0
       0 0101 0100 0011 0011
+      0 0111 0000 1001 1000
---------------------------
         
```

No more overflow and the answer is correct now.

44

Solution (b):

No overflow because using larger bit representation.

b) Assuming that the system uses a doubleword sized number for:

→ 00005433h + 00007098h

```
(Carry)  0   0
  0000 0000 0000 0000 0101 0100 0011 0011
+ 0000 0000 0000 0000 0111 0000 1001 1000
  ---------------------------------------
  0000 0000 0000 0000 1100 0100 1100 1011
```

ZF : ☐
SF : ☐
CF : ☐
OF : ☐
AF : ☐
PF : ☐

45

```
EAX=770CC4CB  EBX=7FFD7098  ECX=00000000  EDX=00401005
ESI=00000000  EDI=00000000  EBP=0012FF94  ESP=0012FF8C
EIP=00401025  EFL=00000A82  CF=0  SF=1  ZF=0  OF=1
```

Figure: The *DumpRegs* if use `word` sized numbers - Example 3(a)

```
EAX=0000C4CB  EBX=00007098  ECX=00000000  EDX=00401005
ESI=00000000  EDI=00000000  EBP=0012FF94  ESP=0012FF8C
EIP=00401022  EFL=00000202  CF=0  SF=0  ZF=0  OF=0
```

Figure: The *DumpRegs* if use `dword` sized numbers - Example 3(b)

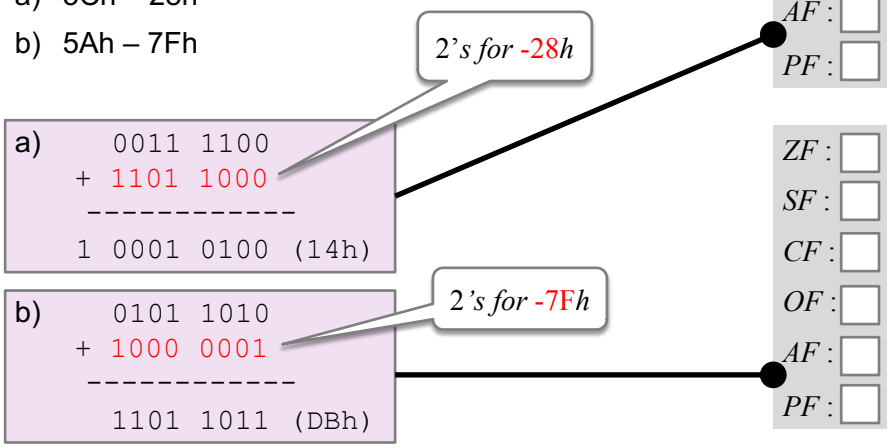
46

Example 4 : Flags

Assuming that the system uses 8-bit system for:

a) 3Ch – 28h

b) 5Ah – 7Fh



47

Design Issues of the Control & Status Register Organization

5

1) **Operating system support.**
   - Certain types of control information are of specific utility to the operating system.
   - The register organization need to some extent to be tailored to a particular operating system .

2) **The allocation of control information between registers and memory.**
   - The designer must decide how much control information should be in registers and how much in memory.
   - The usual trade-off of _____ versus _____ arises.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.494. 48

Module 5a
Central Processing Unit (CPU)

49

5.1 Processor Organization
5.2 Register Organization
5.3 Instruction Cycles
- Overview
- The Indirect Cycle
- Data Flow

5.4 Instruction Pipelining
5.5 Control Unit Operation
5.6 Microprogrammed Control
5.7 Summary

5 Overview

- The processing required for a single instruction is called an **instruction cycle**.
- The instruction processing consists of **three (3) stages** :

_____ *cycle*	- Fetch the instruction; - Decode it; - Fetch operands, if required (indirect).
_____ *cycle*	- Perform the operation; - Store results, if required.
_____ *cycle*	- Recognize pending interrupts.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.85. 50

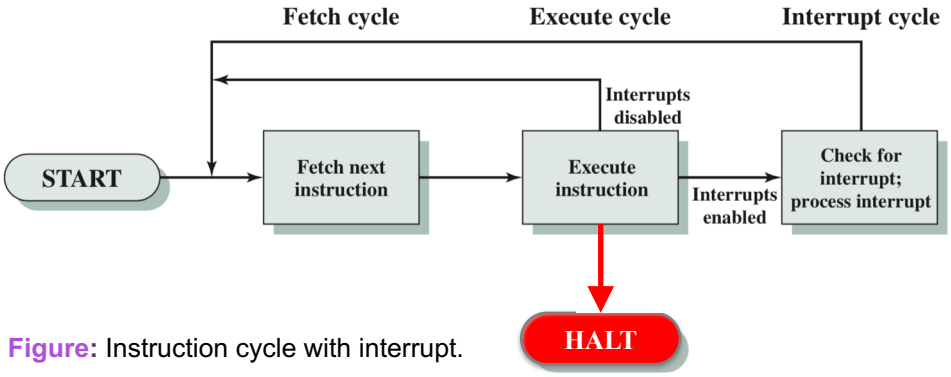


Figure: Instruction cycle with interrupt.

- Program execution halts only if :
  - the machine is turned off,
  - some sort of unrecoverable error occurs, or
  - a program instruction that halts the computer is encountered.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.92. 51

5 The Indirect Cycle

- We can think that the fetching of indirect addresses → another instruction stages.
- The main line of activity consists of alternating instruction fetch and instruction execution activities.
- Following execution, an interrupt may be processed before the next instruction fetch.

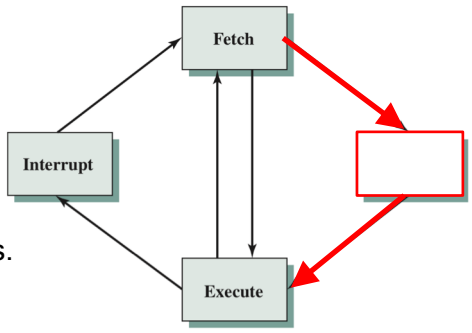


Figure: The instruction cycle.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.497-498. 52

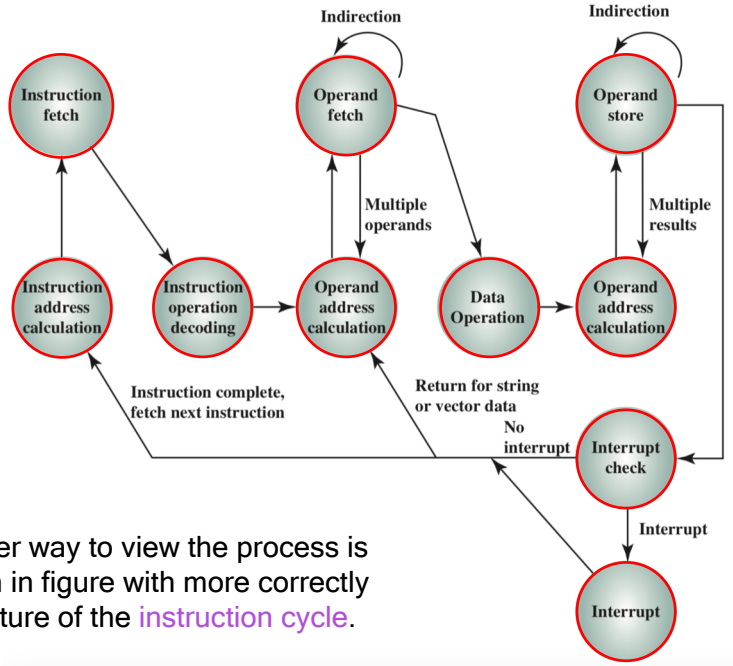


Figure: Instruction cycle state diagram.

- Another way to view the process is shown in figure with more correctly the nature of the instruction cycle.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.497. 53

5 Data Flow

- The exact sequence of events during an instruction cycle depends on the design of the processor.

Let us assume that a processor employs:

Memory Address Register (MAR)
Memory Buffer Register (MBR)
Program Counter (PC)
Instruction Register (IR).

- Each of the smaller cycles consists of a series of steps that involves the processor registers → _____.

Detail in sub-topic: 5.5 and 5.6

- Conventional format using *Register Transfer Language* (RTL)
- **Example**: IP ← IP +1
  PC ← PC +1

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.497, p.709 54

(a) Fetch Cycle

- An instruction is read from memory.

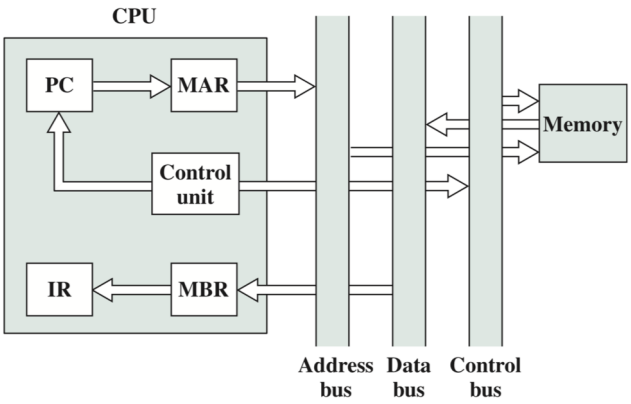


Figure: Data flow, fetch cycle.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.497. 55

RTL (Register Transfer Language)

MAR (Memory Address Register)
MBR (Memory Buffer Register)
PC (Program Counter)
IR (Instruction Register)

- The *fetch cycle* can be written symbolically using RTL as:

t_1: MAR ← IP/PC
t_2: MBR ← [MAR];
IP/PC ← IP/PC + 1
t_3: IR ← MBR

t_1, t_2, t_3 are the timing sequence

56

- The PC contains the address.
- Content of IP/PC is loaded to MAR.
- Then transferred to memory on the address bus.
- Next, CU will activate the read signal of the memory.

- Instruction is accessed from the memory and transfer to CPU on the data bus.
- This instruction address will be loaded into MBR.

- At the same time, IP/PC will be incremented (point to the next instruction).

CU (Control Unit)
MAR (Memory Address Register)
MBR (Memory Buffer Register)
PC (Program Counter)
IR (Instruction Register)

5

t_1: MAR ← IP/PC
t_2: MBR ← [MAR];
IP/PC ← IP/PC + 1
t_3: IR ← MBR

- Instruction in MBR is loaded into IR.
- The opcode of the instruction will be decoded and translated so as to determine the *micro-operations* for the particular instruction.

57

MAR (Memory Address Register)
MBR (Memory Buffer Register)
IR (Instruction Register)

5

(a) Fetch Cycle: *Indirect Cycle*

- Once the *fetch cycle* is over, the control unit examines the contents of the IR to determine if it contains an operand specifier using indirect addressing.

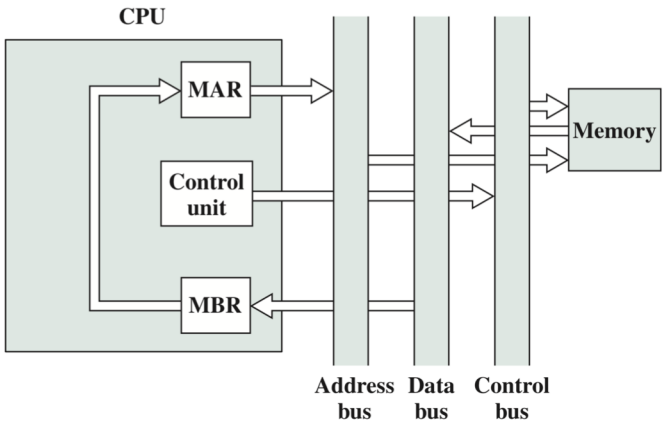


Figure: Data flow, indirect cycle.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.497. 58

RTL (Register Transfer Language)

MAR (Memory Address Register)
MBR (Memory Buffer Register)
IR (Instruction Register)

5

- The *fetch cycle (Indirect addressing)* can be written symbolically using RTL as:

t_1: MAR ← IR[Operand/Address]
t_2: MBR ← Mem[MAR];
t_3: IR[operand] ← MBR

t_1, t_2 , t_3 are the timing sequence

59

CU (Control Unit)

MAR (Memory Address Register)
MBR (Memory Buffer Register)
IR (Instruction Register)

5

- The IR contains the indirect address that loaded to MAR.
- Then transferred to memory on the address bus.
- Next, CU will activate the read signal of the memory.

t_1: MAR ← IR[Operand/Address]
t_2: MBR ← Mem[MAR];
t_3: IR[operand] ← MBR

- Data (operand) is accessed from the memory and transfer to CPU on the data bus.
- This data will be loaded into MBR.

- Data in MBR is loaded into IR.
- The opcode of the instruction will be decoded and translated so as to determine the *micro-operations* for the particular instruction.

60

5

Example 5: Indirect cycle

```
ADD AX,[2400]
```

Involve with fetching data from memory before initiating the *execution cycle*.

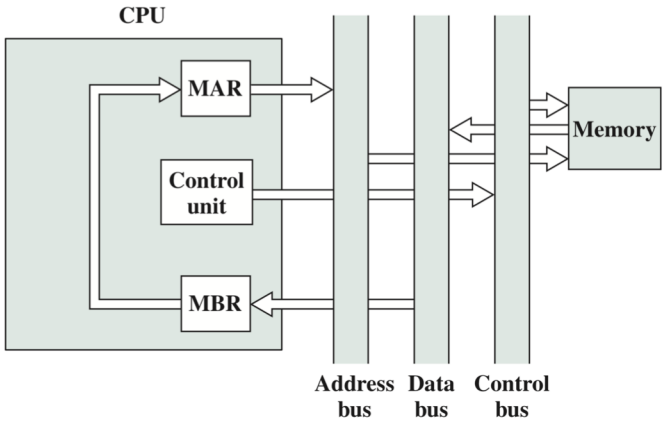


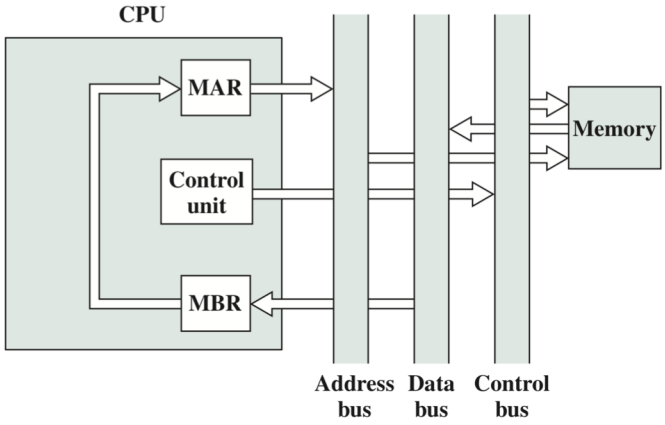
Figure: Data flow, indirect cycle.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.497. 61

5

(1) The control unit request a memory read,

(2) and the result is placed onto the data bus



(5) then placed on the address bus

(4) The address is moved into MAR,

(3) Then copied into MBR

MAR (Memory Address Register)
MBR (Memory Buffer Register)

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.497. 62

CPU (Control Processing Unit)
I/O (Input/Output)
IR (Instruction Register)

5

(b) Execute Cycle

- The _____ of the instruction in IR will be decoded and related control signals will be generated.

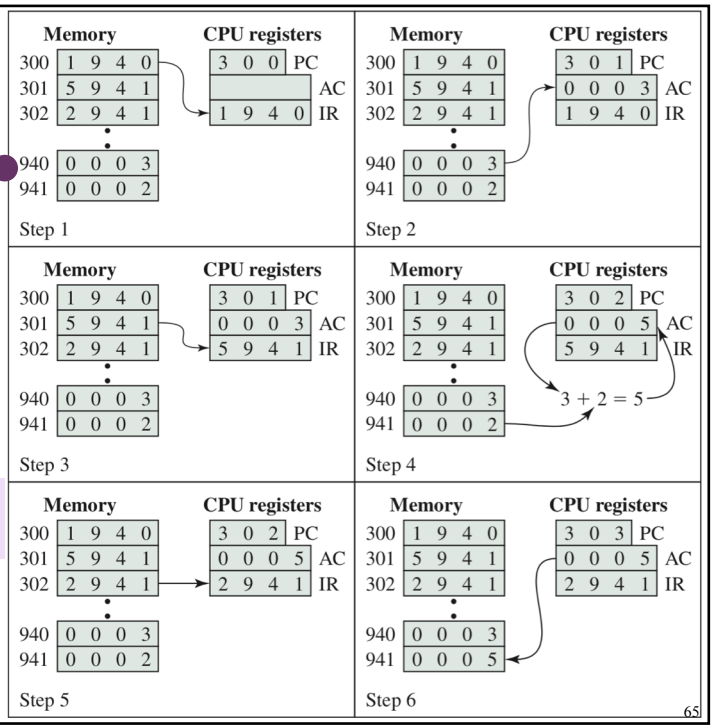
Type of Opcode	Operation
Processor-memory	Data transfer between CPU and main memory.
Processor I/O	Data transfer between CPU and I/O module.
Data processing	Some arithmetic or logical operation on data.
Control	Alteration of sequence of operations. *e.g.* jump Combination of above

63

Example 6:

Example of program execution (contents of memory and registers in hexadecimal)

[940]+[941] → [941]



65

Summary

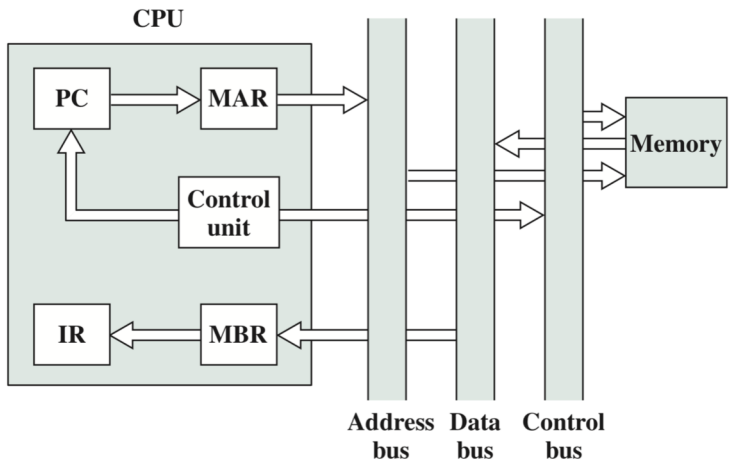
[940]+[941] → [941]

1. The PC contains 300, the address of the first instruction. This instruction (the value 1940 in hexadecimal) is loaded into the instruction register IP and the PC is incremented.
2. The first 4 bits in the IR indicate that the AC is to be loaded. The remaining 12 bits specify the address (940) from which data are to be loaded.
3. The next instruction (5941) is fetched from location 301 and the PC is incremented.
4. The old contents of the AC and the contents of location 941 are added and the result is stored in the AC.
5. The next instruction (2941) is fetched from location 302 and the PC is incremented.
6. The contents of the AC are stored in location 941.

66

Example 7a: Fetch cycle

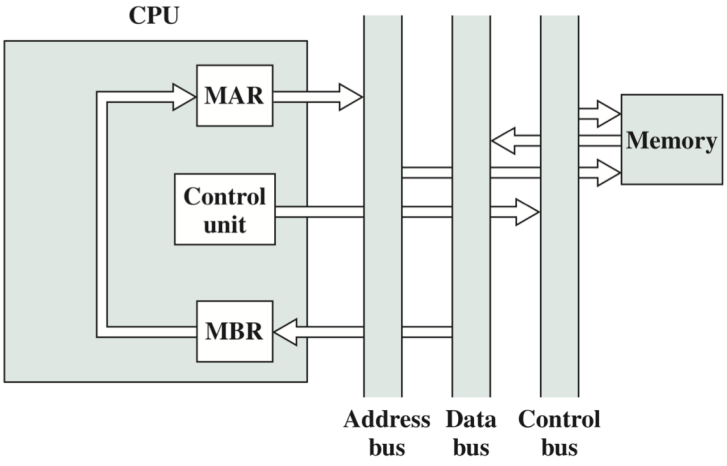
```
MOV AX, NUM
```



William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.497. 67

Example 7b: Indirect cycle

```
MOV AX, NUM
```



William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.497. 68

Example 7c:

`MOV AX, NUM`

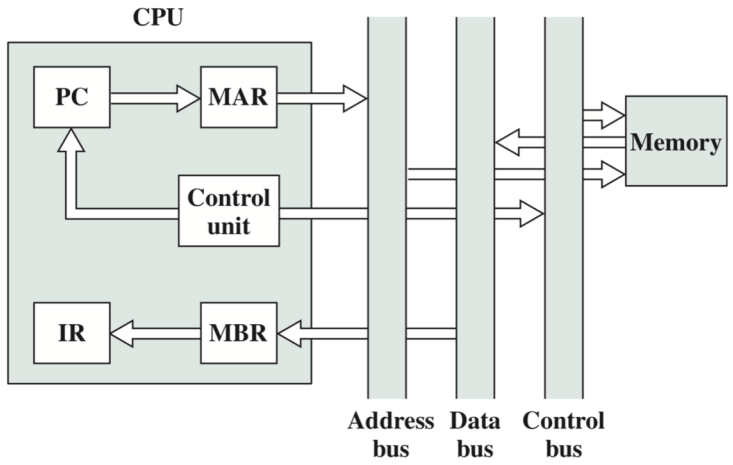
100	MOV AX, [450]
XXX	ZZZZ
450	000A
YYY	BBBB

Clock	IP/PC	MAR	MBR	IR	AX	Micro-operation
t_0	.....					IP/PC = 100
t_1		.....				*Fetch cycle*: MAR ← IP/PC;
t_2	.....		.....			MBR ← [MAR]; IP/PC ← IP/PC +1;
t_3				.....		IR ← MBR *Indirect cycle*;
t_4		.....				MAR← IR[operand/address];
t_5			.....			MBR ← Mem[MAR]; *Execution cycle*:
t_6					.....	AX ← MBR;

69

5

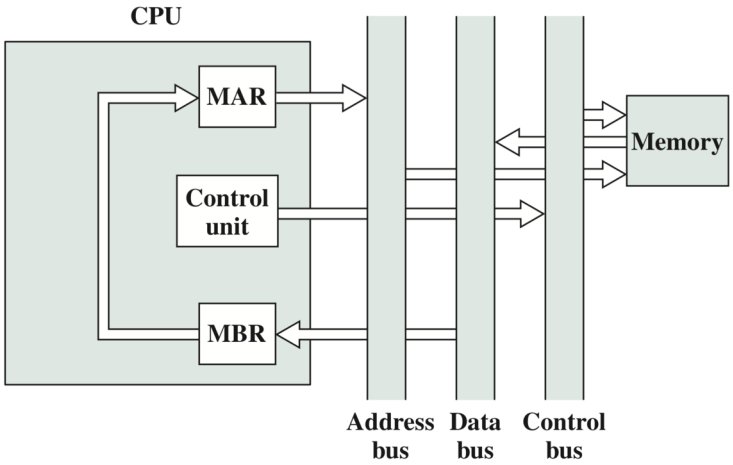
Example 8: Trace the execution of the instruction.



71

5

(Indirect addressing)



72

(c) Interrupt Cycle

5

- All computers provide a way of interrupting the fetch-decode-execute cycle.
- Interrupts occur when:
  - A user break (*e.g.*, Control+C) is issued.
  - I/O is requested by the user or a program.
  - A critical error occurs.
- Interrupts can be caused by hardware or software.

  → Software interrupts are also called _____.

73

5

- Interrupt processing involves adding another step to the fetch-decode-execute cycle as shown below:

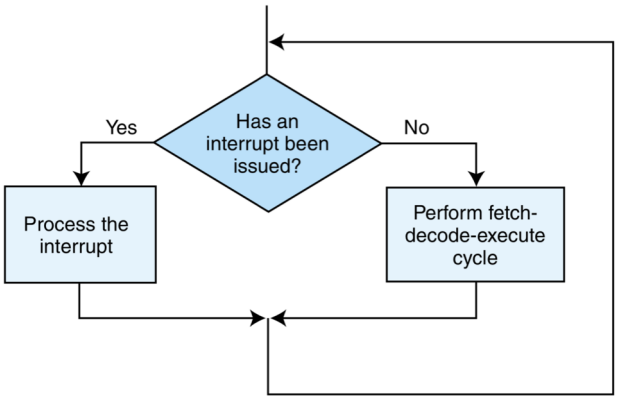


Figure: Modified Instruction Cycle to Check for Interrupt

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture*. United States: Jones and Bartlett Publishers. p.168 74

5

- The current contents of the PC must be saved so that the processor can resume normal activity after the interrupt.

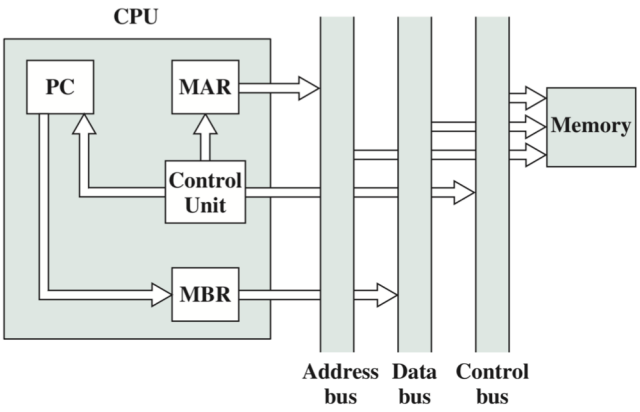


Figure: Data flow, interrupt cycle.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.497. 75

5

Interrupt

- Virtually all computers provide a mechanism by which other modules (I/O, memory) may *interrupt* the normal processing of the processor *.
- Interrupts let the CPU execute its normal instruction sequence and pause to service the external devices only when they signal (the interrupts) that they are ready for the CPU's attention.

* William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.89. 77

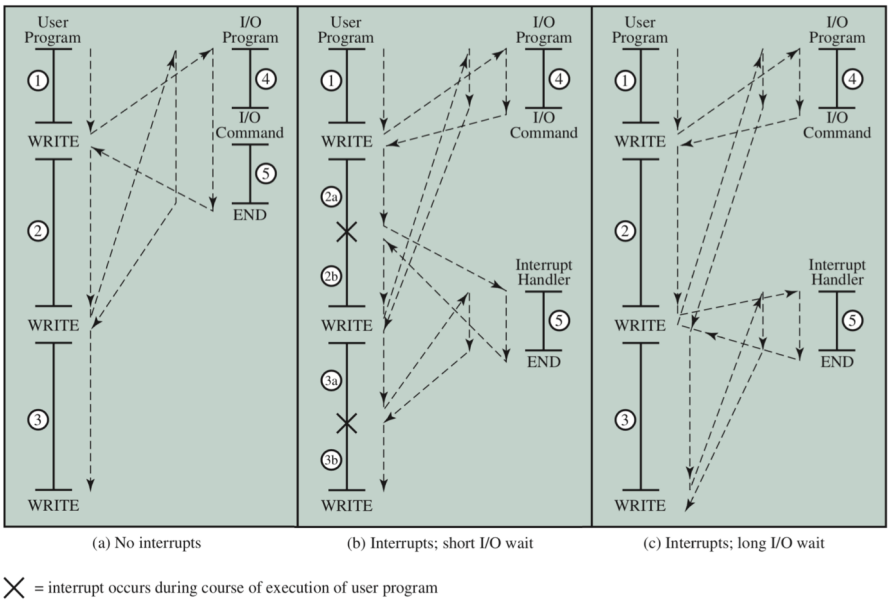


Figure: Program Flow of Control without and with Interrupts

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.90. 78

- To appreciate the gain in efficiency, consider the timing diagram based on the flow of control in previous figures.

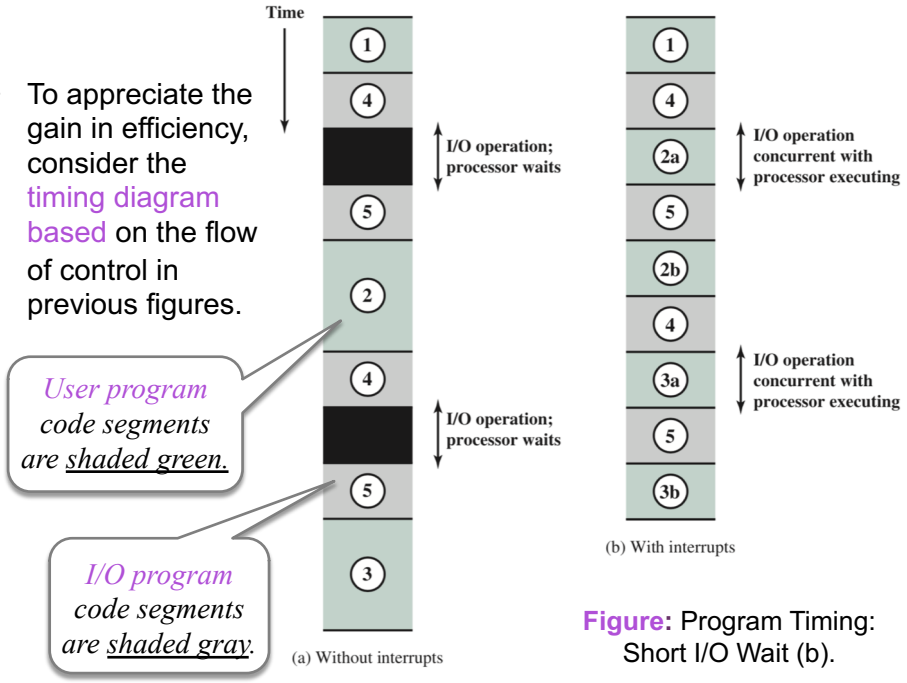


Figure: Program Timing: Short I/O Wait (b).

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.93. 79

- The more typical case, especially for a slow device such as a printer → the I/O operation will take much more time than executing a sequence of user instructions.

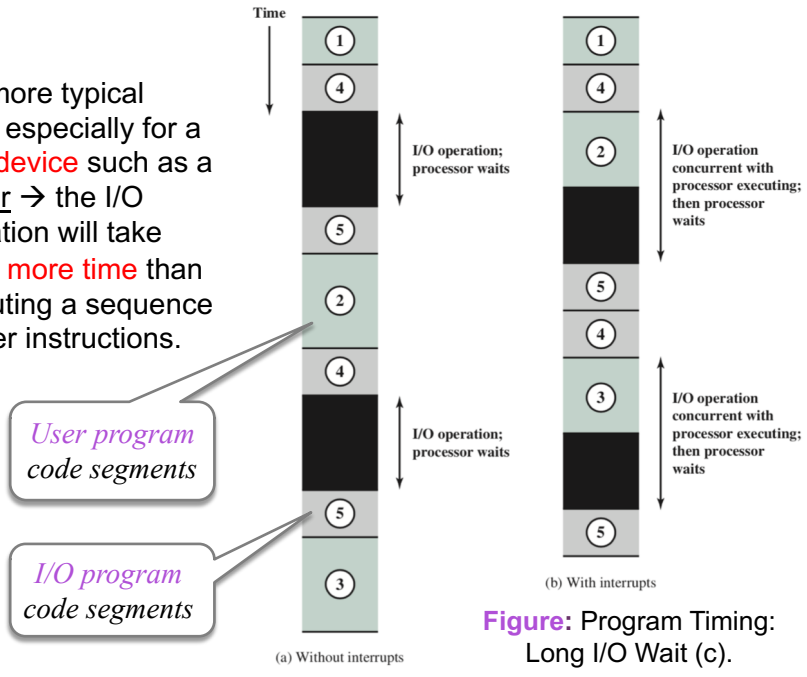


Figure: Program Timing: Long I/O Wait (c).

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.94. 80

5

Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, or reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation, request service from the processor, or to signal a variety of error conditions.
Hardware failure	Generated by a failure such as power failure or memory parity error.

Figure: Class of interrupts.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.93. 81

5

Summary 1

- The major components of a computer system are its *control unit*, *registers*, *memory*, *ALU*, and *data path*.
- Computers run programs through iterative *fetch-decode-execute* cycles → Summarize the instruction cycle.
- Distinguish between user-visible and control/status registers, and discuss the purposes of *registers* in each category.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.489. 82

Module 5b
Central Processing Unit (CPU)

5.1 Processor Organization
5.2 Register Organization
5.3 Instruction Cycles

5.4 Instruction Pipelining

- Overview
- Pipeline Strategy
- Pipeline Performance
- Pipeline Hazards (Limitations)

5.5 Control Unit Operation
5.6 Microprogrammed Control
5.7 Summary

5 Overview

- Organizational enhancements to the processor can improve performance another approach → _____.*
- CPU break *fetch-decode-execute* cycle into tasks that performed in parallel so that more than one tasks can be executed at a time.

- The concept of pipelining is to allow the processing of a new task even though the processing of previous task has not ended.

* William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.500. 84

Analogy: Pipeline Laundry

- Anyone who has done a lot of laundry has intuitively used pipelining.
- The non-pipelined approach to laundry would be as follows:

1. Place one dirty load of clothes in the washer.
2. When the washer is finished, place the wet load in the dryer.
3. When the dryer is finished, place the dry load on a table and fold.
4. When folding is finished, ask your roommate to put the clothes away.

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5th Edition)*. United States: Elsevier, p.272. 85

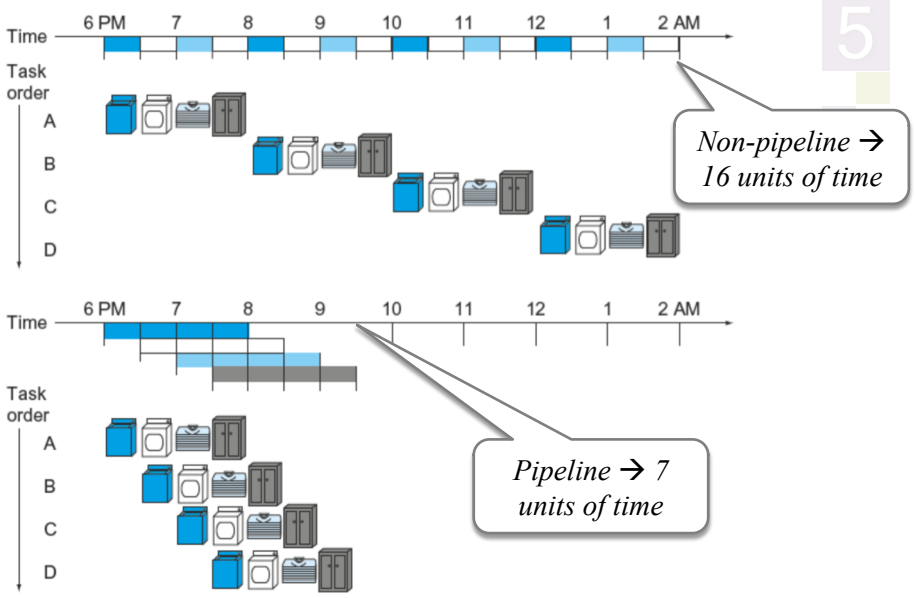


Figure: The laundry analogy (4 loads: A, B, C, D) for pipelining.

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5th Edition)*. United States: Elsevier, p.273. 86

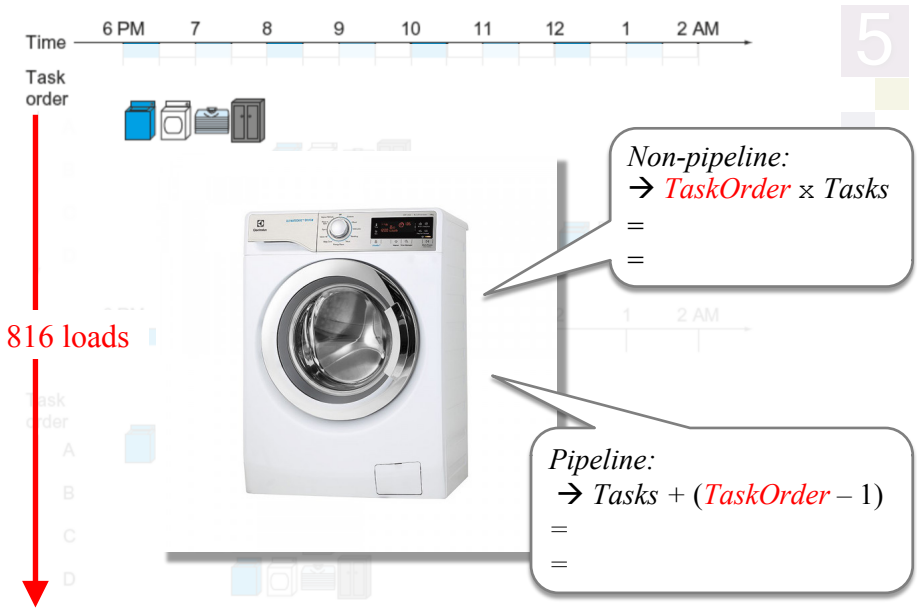
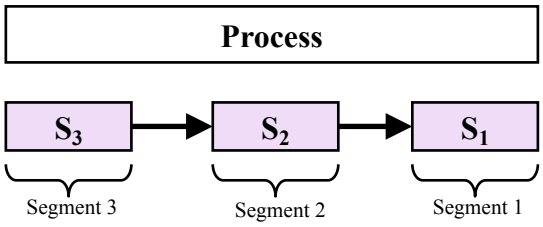


Figure: The laundry analogy (816 loads) for pipelining.

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5th Edition)*. United States: Elsevier, p.273. 87

Instruction Pipeline



- A single instruction cycle process is divided into several small independent tasks (or _____).
- Some segments can be performed in parallel.

With the use of pipeline, the average CPU is reduced.

88

1) Data operands pass through all segments (S) in sequence.
2) Each segment consists of a circuit that performs sub-operation.
3) Segments are separated by registers (R), that hold the intermediate results between stages.

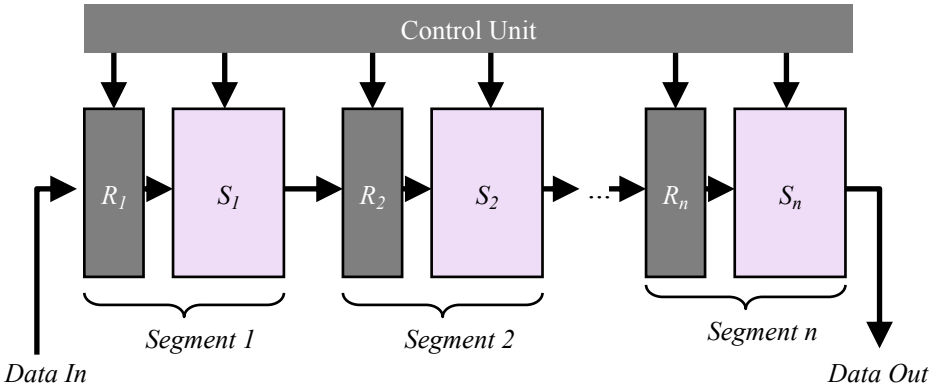
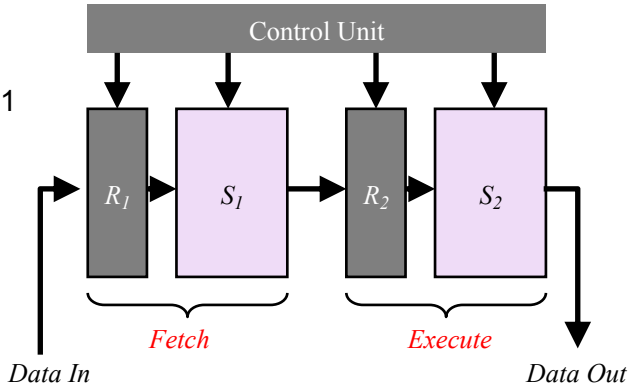


Figure: CPU pipeline structure.

89

Example 9: Case 1



Pipeline

Fetch #1	Execute #1
Fetch #2	Execute #2

90

5 Pipeline Strategy

- The process is referred to as pipelining when a new inputs are accepted at one end before previously accepted inputs appear as outputs at the other end.
- As a simple approach, consider subdividing instruction processing into two stages:
  - _____ instruction and
  - _____ instruction.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.500. 91

Instruction prefetch or fetch overlap;

This process will speed up instruction execution.

- There are times during the execution of an instruction when main memory is not being accessed.
- This time could be used to fetch the next instruction in _____ with the execution of the current one.

Figure: Two-stage instruction pipeline: *simplified view*

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.500. 92

5

- This doubling of execution rate is unlikely for two reasons:

1) The execution time will generally be longer than the fetch time.
2) A conditional branch instruction makes the address of the next instruction to be fetched unknown.

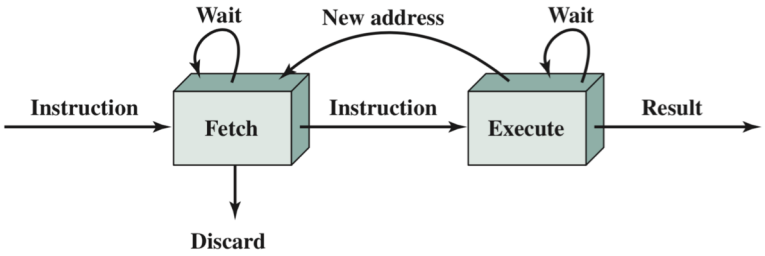


Figure: Two-stage instruction pipeline: *expanded view*

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.501. 93

- To gain further speedup, the pipeline must have more stages.
- Let consider the following decomposition of the instruction processing:

- ❑ _____ (FI): Read next expected instruction into buffer.
- ❑ _____ (DI): Determine the opcode and the operand specifiers.
- ❑ _____ (CO): Calculate the effective address of each source operand.
- ❑ _____ (FO): Fetch each operand from memory.
- ❑ _____ (EI): Perform the indicated operation and store the result.
- ❑ _____ (WO): Store the result in memory.

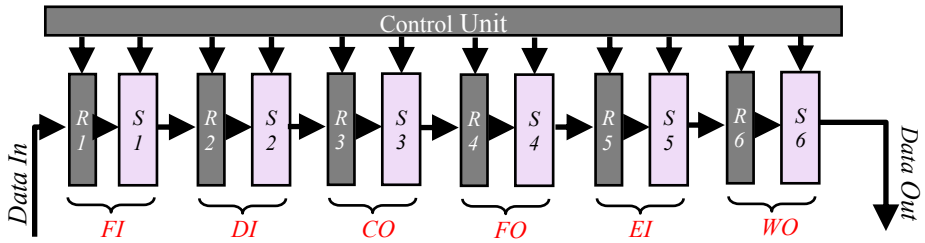
William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.501. 94

5

Example 10: Case 2

To implement the instruction cycle with multiple stages in simpler form:

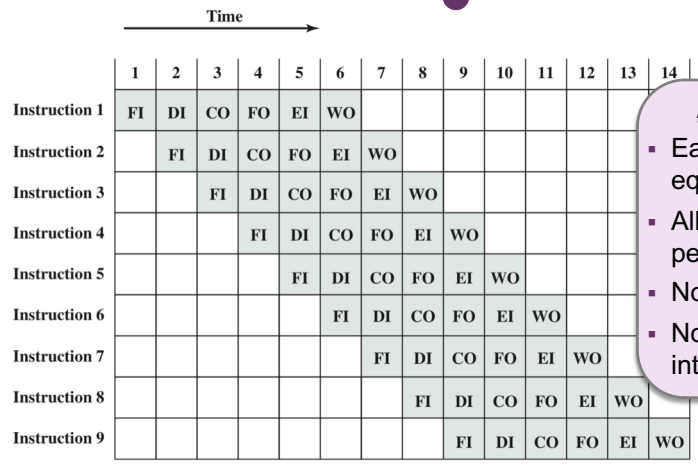
- Use multiple execution "functional units" to parallelize the actual execution phase of several instructions.



95

Example 11:

Figure shows that a 6 stages pipeline can reduce the execution time for 9 instructions from 54 time units to 14 time units.



Assumptions:

- Each stages → equal duration.
- All stages can be performed in parallel.
- No memory conflicts.
- No branching or interrupt happen.

Figure: Timing Diagram for Instruction Pipeline Operation

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.502. 96

5 Pipeline Performance

- Here some simple measures of pipeline performance and relative speedup, S.
- Let the execution time, T, required for a non-pipeline and pipeline respectively, and pipeline with k stages to execute n instructions as:

Non-pipeline:
t_n → the cycle time

$$T_n = t_n \times n$$

Pipeline:
t_p → the cycle time

$$T_p = t_p(k + (n-1))$$

Note: $k = 1$

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.504. 97

k (Stages)
n (# instructions)
T (Execution time)
t (cycle time)

5

- The speedup, S, for the instruction pipeline compared to execution without the pipeline is defined as:

$$S = \frac{T_n}{T_p} = \frac{t_n \times n}{t_p(k + (n-1))}$$

- If n is too large from k, $n >> k$, and when $t_n = k$ x t_p, thus maximum speedup:

$$S_{\max} = k$$

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.504. 98

5

Example 12: Pipeline (Non-ideal case)

Given the 4 segment pipeline whereby each has a delay time as follow:

- Segment 1 : 40*ns*
- Segment 2 : 25*ns*
- Segment 3 : 45*ns*
- Segment 4 : 45*ns*

The delay time for the interface register is 5*ns*. Calculate the:

a) cycle time of the non-pipeline and pipeline.

b) execution time for 100 tasks.

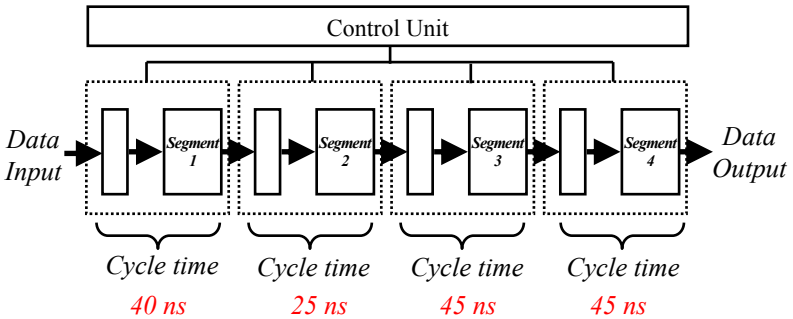
c) real speedup.

d) Maximum speedup.

99

5

Solution :



100

5

a) Cycle time of the non-pipeline and pipeline.

t_n = (Total executions) + interface delay

t_n = [ ] + [ ] = [ ]

t_p = (the longest delay for execution) + interface delay

t_p = [ ] = [ ]

b) Execution time for 100 tasks.

Non-Pipeline:

$T_n = t_n \times n$

= [ ]

= [ ]

Pipeline:

$T_p = t_p(k + (n-1))$

= [ ]

= [ ]

101

5

c) Real speedup.

$$S = \frac{T_n}{T_p}$$

$$= \frac{t_n \times n}{t_p(k + (n-1))}$$

= [ ]

= [ ]

d) Maximum speedup.

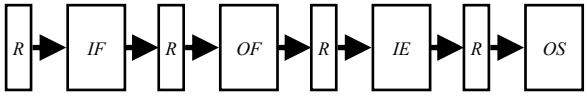
$S_{max} = k$ = [ ]

102

Example 13:

Segment Name	Segment Execution Time (ns)
Instruction Fetch & Decode (IF)	52
Operand Fetch (OF)	40
Instruction Execute (IE)	30
Operand Store (OS)	40

- There are 4 segments as follows:
- The interface delay is 3 *ns*.
- The simple block diagram of the pipeline:



Draw the space time diagram and calculate the:

a) cycle time of the non-pipeline and pipeline.

b) execution time for 50 tasks.

c) real speedup.

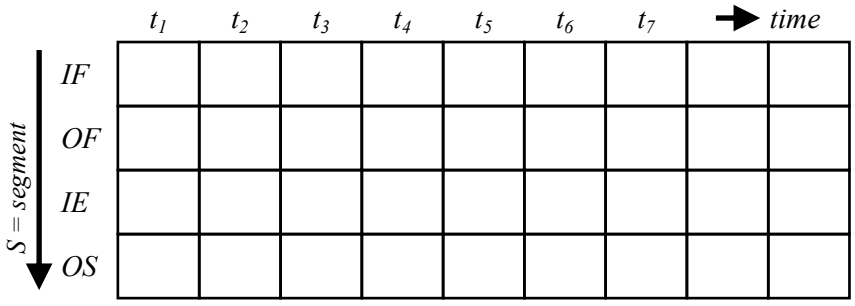
d) Maximum speedup.

103

Solution :

5

- Purpose of diagram – to show the events of a pipeline
- Assume only 4 instructions.



104

5

a) cycle time of the non-pipeline and pipeline.

t_n = (Total executions) + interface delay

$t_n =$ [ ] + [ ] = [ ]

t_p = (the longest delay for execution) + interface delay

$t_p =$ [ ] = [ ]

b) execution time for 50 tasks.

Non-Pipeline:

$T_n = t_n \times n$

= [ ]

= [ ]

Pipeline:

$T_p = t_p(k + (n-1))$

= [ ]

= [ ]

105

5

c) real speedup. [ ]

d) Maximum speedup.

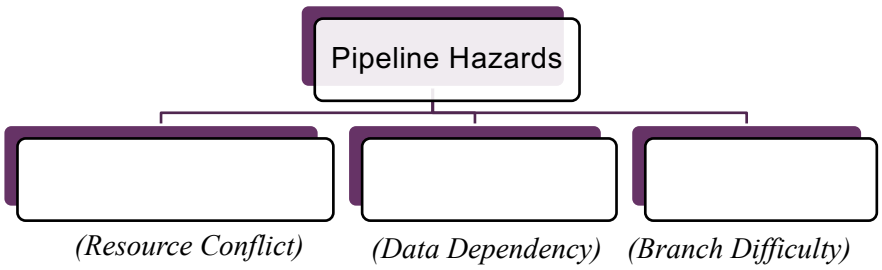
$S_{\max} = k = 4$

106

5

Pipeline Hazards (Limitations)

- A pipeline hazard occurs when the pipeline, or some portion of the pipeline, must stall because conditions do not permit continued execution.
- Such a pipeline stall is also referred to as a pipeline bubble.



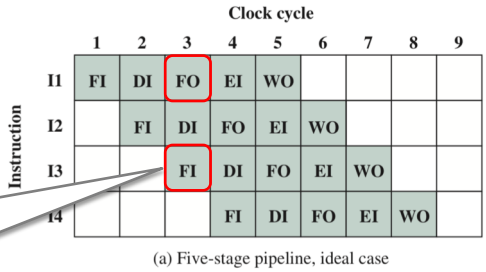
5

(a) Resource Hazards

(Resource Conflict)

- A resource hazard occurs when two (or more) instructions that are already in the pipeline need the same resource.
- The result is that the instructions must be executed in serial rather than parallel for a portion of the pipeline.

We have a conflict with 2 instructions need the same resource (i.e. memory) → data reads / writes can only happen 1 at a time.



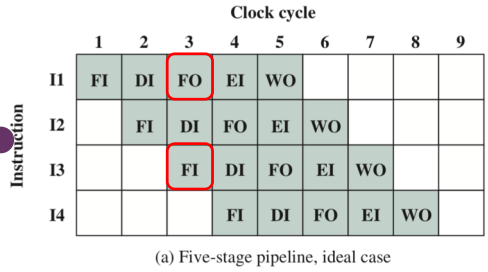
(a) Five-stage pipeline, ideal case

Figure: Example of resource hazards

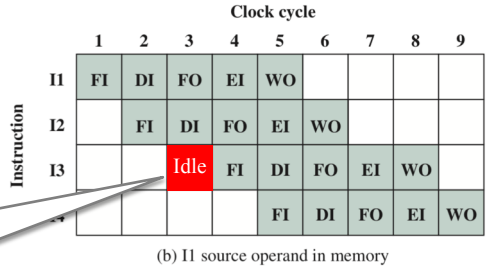
Resource Hazards

Solution:

Fetch Instruction (FI)
Decode Instruction (DI)
Calculate Operands (CO)
Fetch Operands (FO)
Execute Instruction (EI)
Write (store) Operand (WO)



(a) Five-stage pipeline, ideal case

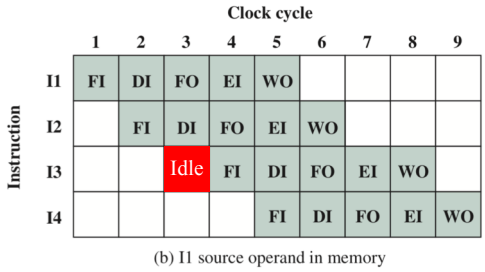


(b) I1 source operand in memory

Source operand for instruction I1 is in memory, rather than a register → idle for one cycle (wasted)

Figure: Example of resource hazards

- Conflict of resources introduce '*pipeline delay*'.
- A delay in any stage can cause pipeline stalls.



(b) I1 source operand in memory

Figure: Example of resource hazards

(b) Data Hazards

(Data Dependency)

Example:

```
ADD  EAX, EBX   /* EAX=EAX+EBX
SUB  ECX, EAX   /* ECX=ECX–EAX
```

- A data hazard occurs when there is a conflict in the access of an operand location.
- Two instructions in a program are to be executed in sequence and both access a particular memory or register operand.

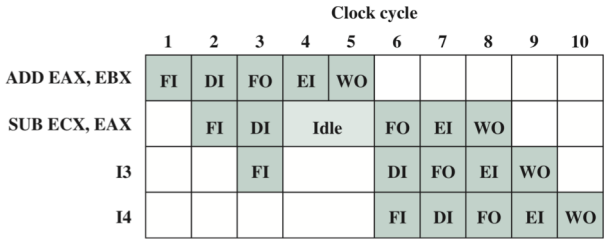


Figure: Example of data hazards

Data Hazards

Example:

```
ADD  EAX, EBX   /* EAX=EAX+EBX
SUB  ECX, EAX   /* ECX=ECX–EAX
```

The ADD *instruction does not update register* EAX *until the end of stage* 5.

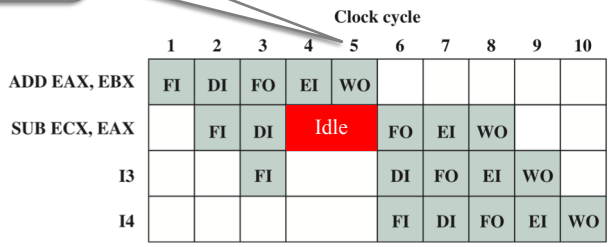


Figure: Example of data hazards

Data Hazards : Solutions

5

Hardware interlocks:

- An interlock is a circuit that detects instructions with data dependency and inserts required delays to resolve conflicts.

Operand forwarding:

- Allowing the result of ALU to be used by other ALU operations in the next instruction cycle.

Delayed load:

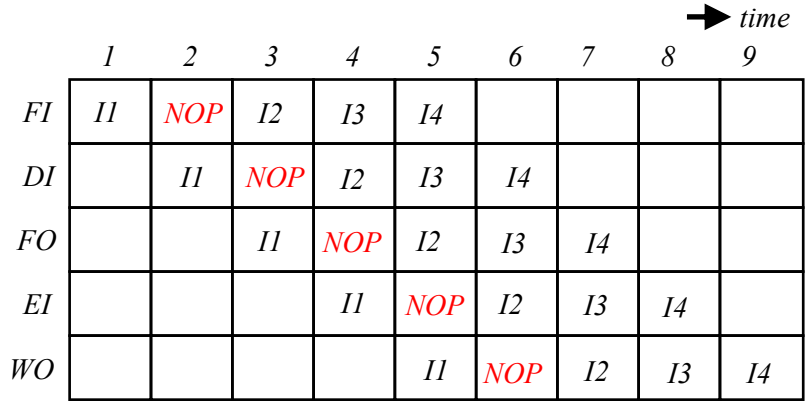
- Compiler is used to detect conflict and reorder instructions to delay loading of conflicting data.
- Using _____ instruction.

113

Example 14 :

I1: ADD EAX, EBX
NOP
I2: SUB ECX, EAX
I3:
I4:

Solution of data hazard using Delayed Load → NOP



114

Example 15 :

I1: ADD AH, BH
I2: SUB AH, 2
I3: CMP AH, 0

Consider the instructions given.

Where is/are the problem(s) with the pipeline implementation? Briefly justify your answer.

Data hazard → *Both are about to access the same register* AH.

	1	2	3	4	5	6	7		
FI	*I1*	*I2*	*I3*						
DI		*I1*	*I2*	*I3*					
FO			*I1*	*I2*	*I3*				
EI				*I1*	*I2*	*I3*			
WO					*I1*	*I2*	*I3*		

115

Solution :

I1: ADD AH, BH
NOP
I2: SUB AH, 2
NOP
I3: CMP AH, 0

Solution of data hazard using Delayed Load → NOP

time →

	1	2	3	4	5	6	7	8	9
FI	*I1*	*NOP*	*I2*	*NOP*	*I3*				
DI		*I1*	*NOP*	*I2*	*NOP*	*I3*			
FO			*I1*	*NOP*	*I2*	*NOP*	*I3*		
EI				.....	.....	.....	.....	.....	
WO					.....	.....	.....	.....	.....

116

(c) Control Hazards

(_____)

- Occurs when the pipeline makes the wrong decision on a branch prediction (change the PC value).
- It brings instructions into the pipeline that must subsequently be discarded.
- One of the major problems in designing an instruction pipeline is assuring a steady flow of instructions to the initial stages of the pipeline.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.507, 509. 117

- For the pipeline to have the desired operational speedup, we must "feed it" with long strings of instructions.
- However, 15-20% of instructions in an assembly-level code are (conditional) branches.
- Of these, 60-70% take the branch to a target address.

- Impact of the branch is that pipeline never really operates at its full capacity:
  → limiting the performance improvement that is derived from the pipeline.

118

Example 16:

Figure shows timing diagram for 9 instruction pipeline operation with 6 stages of pipeline.

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.502. 119

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

The diagram assumes that each instruction goes through all 6 stages of the pipeline. → Not true

e.g. Load instruction don't need WO stage !

Most memory systems will not permit the accesses (FI, FO, WO) occur simultaneously

→ Desired value may be in cache, or the FO or WO stage may be null !

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.502. 120

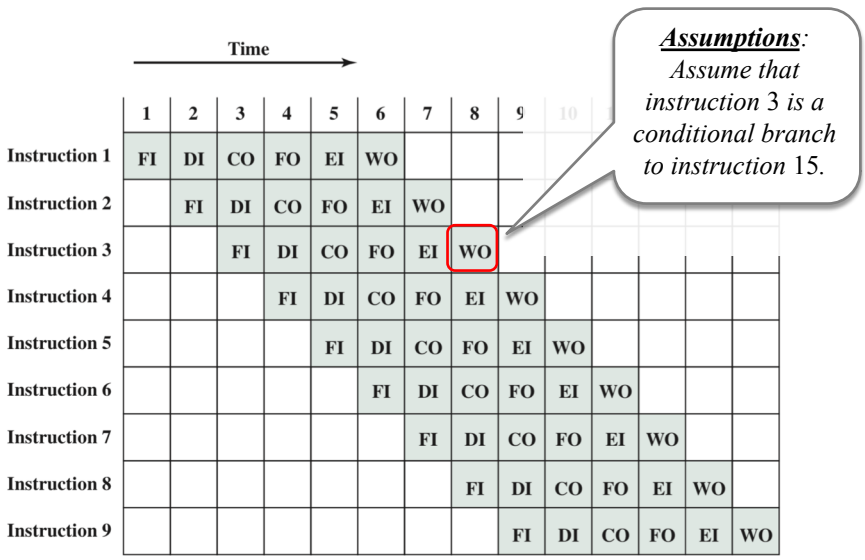


Figure: The Effect of a Conditional Branch on Instruction Pipeline Operation

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.503. 121

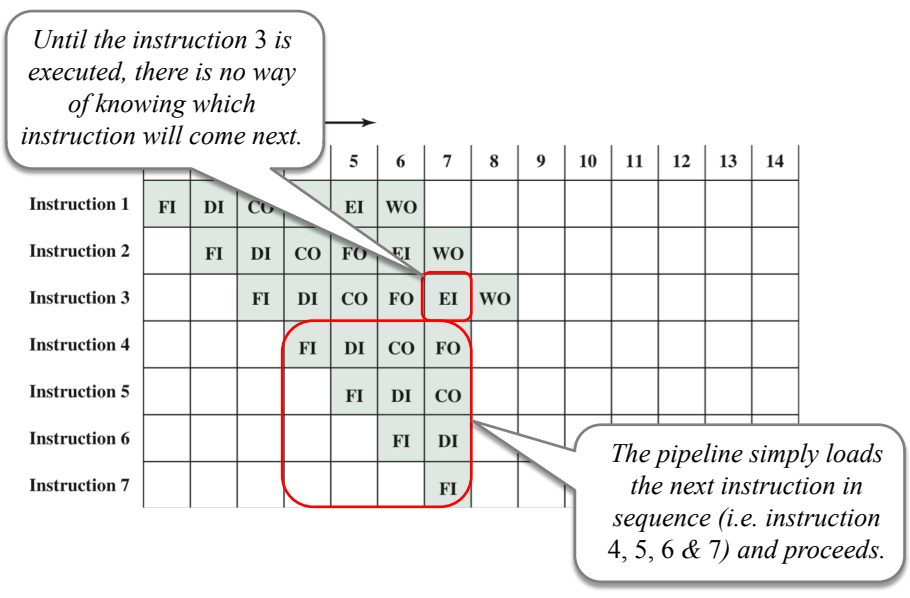


Figure: The Effect of a Conditional Branch on Instruction Pipeline Operation

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.503. 122

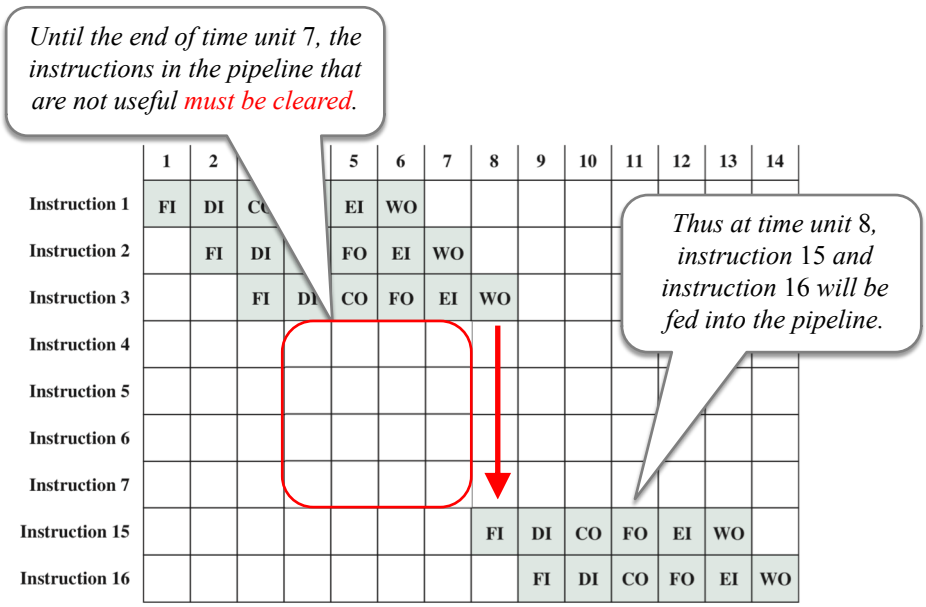


Figure: The Effect of a Conditional Branch on Instruction Pipeline Operation

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.503. 123

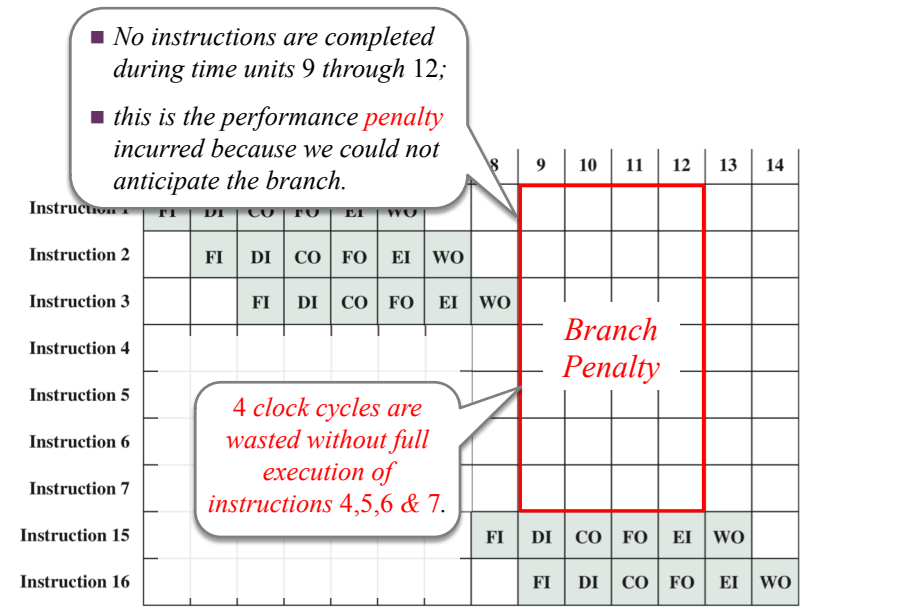
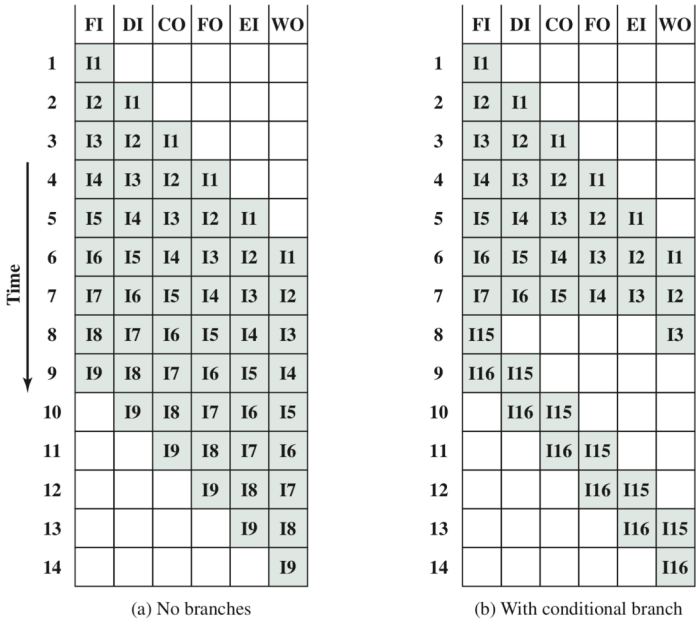


Figure: The Effect of a Conditional Branch on Instruction Pipeline Operation

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.503. 124



(a) No branches (b) With conditional branch

Figure: An Alternative Pipeline Depiction

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.505. 125

Control Hazards : Solutions

- the logic needed for pipelining to account for branches and interrupts.
  - _____ (use of NOP instruction).
  - Rearranging the instructions.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.503. 126

Example 17 : Solution for branching using delay branch (with NOP)

When the compiler detect a branch, it will automatically insert several NOP so that there is no interruptions in the pipeline.
(Assume 4 segments in a pipeline)

```
        MOV
        INC
        ADD
        RET I1
I1:
```

127

Solution :

When the compiler detect a branch, it will automatically insert several NOP so that there is no interruptions in the pipeline.
(Assume 4 segments in a pipeline)

$= No\ of\ segment - 1$

$=$

$=$

```
        MOV
        INC
        ADD
        RET I1
        NOP
        NOP
        NOP
I1:
```

(Add 3 NOP)

128

5

Illustrate with space-time diagram.

Wasted 3 *pipeline clock cycles*

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*
FI	MOV	INC	ADD	RET	NOP	NOP	NOP	I1	I2	I3			
DI		MOV	INC	ADD	RET	NOP	NOP	NOP	I1	I2	I3		
EI			MOV	INC	ADD	RET	NOP	NOP	NOP	I1	I2	I3	
WO													

129

Example 18 : Solution for branching using rearranging instructions.

When the compiler detect a branch, it will rearrange the instructions so that there is no wasted pipeline.

$= No\ of\ segment - 1$

$= k - 1$

$= 4 - 1 = 3$ (move up 3)

```
        MOV
        INC
        ADD
        RET I1
I1:
```

131

5

Illustrate with space-time diagram.

No more wasted pipeline !

	1	2	3	4	5	6	7	8	9	10	11	12	13
FI	RET	MOV	INC	ADD	I1	I2	I3						
DI		RET	MOV	INC	ADD	I1	I2	I3					
EI			RET	MOV	INC	ADD	I1	I2	I3				
WO				.....	.....	.....	.....	.....	.....	.....			

132

Example 19 :

Consider 4 segments involved in a pipeline: *Fetch Instruction* (FI), *Decode Instruction* (DI), *Execute Instruction* (EI) and *Write Operand* (WO).

Based on the instructions given, solve the branching problem using:

(a) delay branch (with NOP).

(b) Rearrange instructions.

```
I1:          MOV   AX, BX
I2:          ADD   BX, CX
I3:          INC   CX
I4:          JMP   LOSS
I5:          MOV   DX, 1
I6:          INC   BX
. . .
I12: LOSS    DEC   AX
```

133

Solution (a):

Solution for branching using delay branch (with NOP)

Total number of NOP to be inserted:

$$= No\ of\ segment - 1$$
$$= k - 1$$
$$= 4 - 1 = 3$$

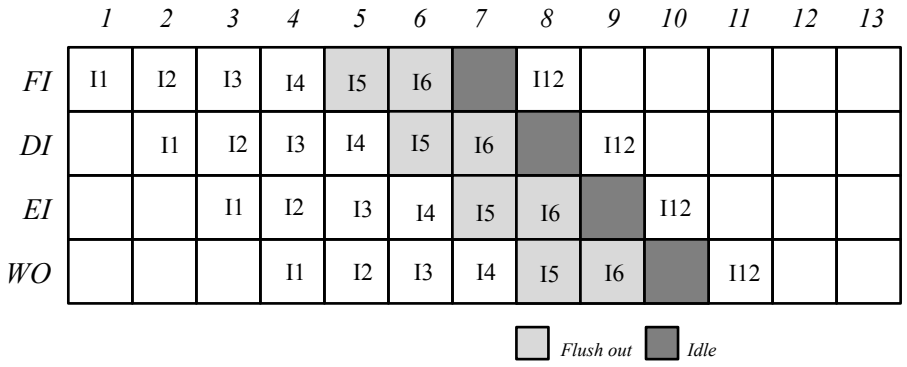
```
I1:          MOV   AX, BX
I2:          ADD   BX, CX
I3:          INC   CX
I4:          JMP   LOSS
             NOP
             NOP
             NOP
I5:          MOV   DX, 1
I6:          INC   BX
. . .
I12: LOSS    DEC   AX
```

(Add 3 NOP)

134

5

Illustrate with space-time diagram for (a).

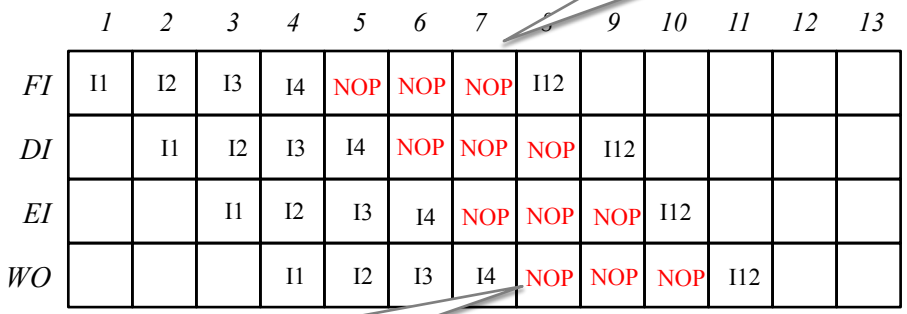


135

5

Illustrate with space-time diagram for (a).

Create 3 *NOP instruction cycle before targeted label.*



Wasted 3 *pipeline clock cycles*

136

Solution (b):

Solution for branching using rearranging instructions.

```
I4:          JMP   LOSS
I1:          MOV   AX, BX
I2:          ADD   BX, CX
I3:          INC   CX
I5:          MOV   DX, 1
I6:          INC   BX
. . .
I12: LOSS    DEC   AX
```

Total number to move up the instruction:

$$= No\ of\ segment - 1$$
$$= k - 1$$
$$= 4 - 1 = 3$$

(move up 3)

137

5

Illustrate with space-time diagram for (b).

No more wasted pipeline !

	1	2	3	4	5	6	7	8	9	10	11	12	13
FI	I4	I1	I2	I3	I12								
DI		I4	I1	I2	I3	I12							
EI			I4	I1	I2	I3	I12						
WO				I4	I1	I2	I3	I12					

138

5 Summary 2

- Instruction pipelining is a powerful technique for enhancing performance but requires careful design to achieve optimum results with reasonable complexity.
- Discussed the principle behind instruction pipelining and how it works in practice.
- Compared and contrasted the various forms of pipeline hazards:
  - _____,
  - _____,
  - _____ or *branch difficulty*.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.489,. 139

Module 5c
Central Processing Unit (CPU)

5

Overview

- Here, we have to know the external interfaces, usually through a bus, and how interrupts are handled.
- Six items might be termed the functional requirements for a processor should do and being controlled :

1. Operations (opcodes)	4. I/O module interface
2. Addressing modes	5. Memory module interface
3. Registers	6. Interrupts

142

5 Micro-Operation (μOP)

- The operation of a computer, in executing a program, consists of a sequence of instruction cycles, with one machine instruction per cycle.
- Each instruction cycle is made up of a number of smaller units.
  - One subdivision that we found convenient is fetch, indirect, execute, and interrupt, with only *fetch* and *execute* cycles always occurring.
- Each of the smaller cycles involves a series of steps, each of which involves the processor registers.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.709. 143

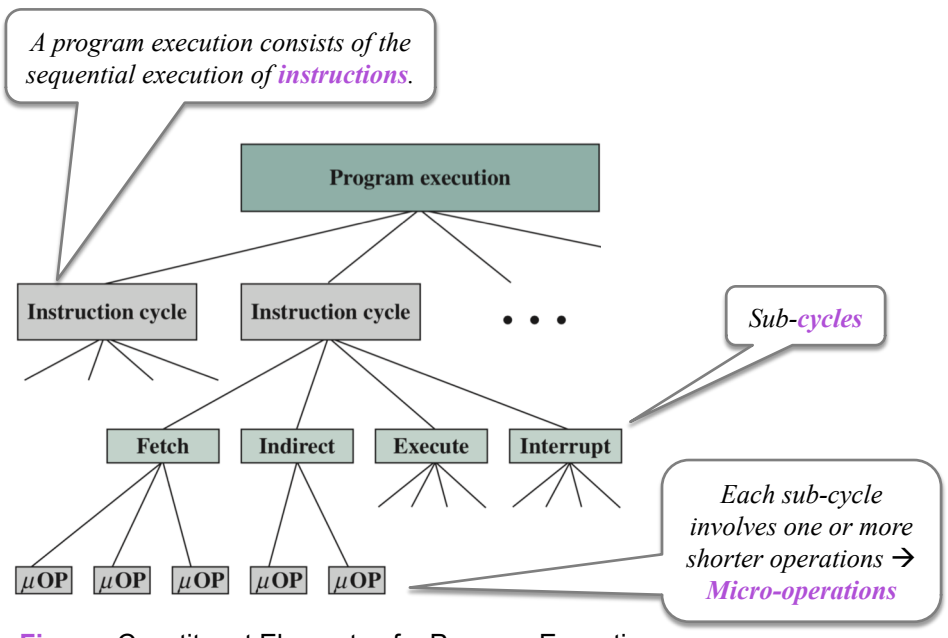


Figure: Constituent Elements of a Program Execution

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.709. 144

5 Control of the Processor

- The basis for the design and implementation of the control unit are based on the definition of a functional requirements.
- Three-step process leads to a characterization of the Control Unit (CU):

1) Define the **basic elements** of the processor.
2) Describe the **micro-operations** (μ**OP**) that the processor performs.
3) Determine the **functions** that the Control Unit (CU) must perform to cause the micro-operations to be performed.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.715. 145

ALU (Arithmetic Logic Unit)
CU (Control Units)
I/O (Input Output)

5 (1) Basic Elements of the Processor

- The basic functional elements of the processor are as follow :

- _____ – the functional essence of the computer.
- _____ – causes operations to happen within the processor.
- _____ – used to store data (also status) internal to the processor.
- Internal data paths – used to move data between registers and between register and ALU.
- External data paths – Link registers to memory and I/O modules, often by means of a system bus.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.715. 146

(2) Micro-Operation of that Processor Performs

5

- All micro-operations (μ**OP**) fall into one of the following categories :

  - Transfer data from one _____ to another _____.
  - Transfer data from a register to an external interface (*e.g.* system bus).
  - Transfer data from an external interface to a register.
  - Perform an arithmetic or logic operation, using registers for input and output.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.715. 147

(3) Control Unit Functions

This is a functional description of what the Control Unit (CU) does → achieved through the use of control signals

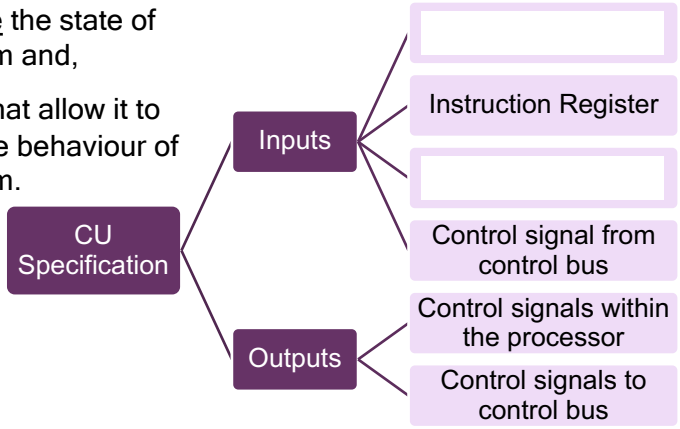
Tasks of CU

	Execution
The CU causes the processor to step through a series of micro-operations in the proper sequence, based on the program being executed.	The CU causes each micro-operation to be performed.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.715. 148

Control Signals

- For the Control Unit (CU) to perform its function, it must have:
  - *Inputs* that allow it to determine the state of the system and,
  - *Outputs* that allow it to control the behaviour of the system.



William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.716-717. 149

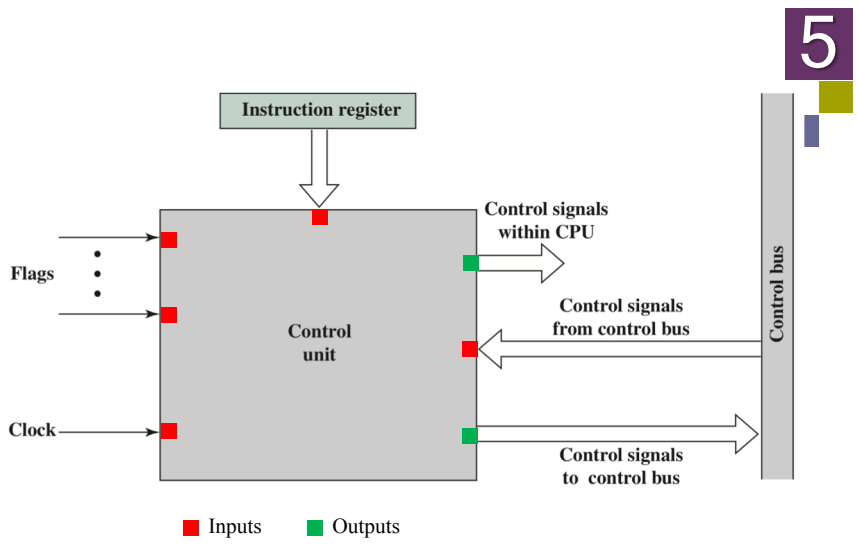


Figure: Block Diagram of the Control Unit

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.716. 150

5

Table: The inputs of CU specification.

Inputs	Description
Clock	This is how the Control Unit (CU) "keeps time."
Instruction Register	The opcode and addressing mode of the current instruction are used to determine which micro-operations to perform during the execute cycle.
.....	These are needed by the CU to determine the status of the processor and the outcome of previous ALU operations.
Control signal from control bus	The control bus portion of the system bus provides signals to the CU.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.716-717. 151

5

Table: The outputs of CU specification.

Outputs	Description
Control signal within the processor	These are two types: ❑ those that cause data to be moved from one register to another, and ❑ those that activate specific ALU functions
Control signals to control bus	These are also of two types: ❑ control signals to memory, and ❑ control signals to the I/O modules.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.716-717. 152

MBR (Memory Buffer Register) | MAR (Memory Address Register)
PC (Program Counter)
IR (Instruction Register)
AC (Accumulator)
ALU (Arithmetic Logic Unit)

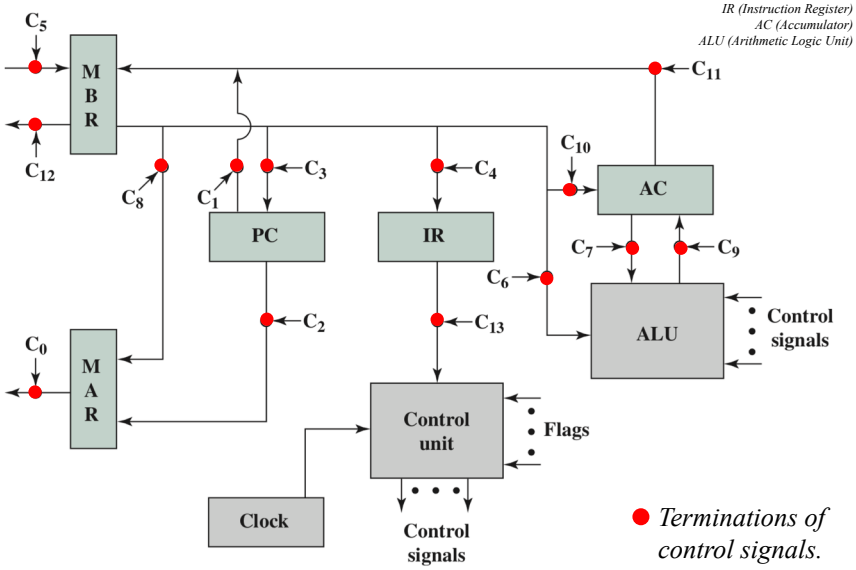


Figure: Data paths and control signals.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.718. 153

MBR (Memory Buffer Register) | MAR (Memory Address Register)
PC (Program Counter)
IR (Instruction Register)

	Micro-operations	Active Control Signals
Fetch:	t_1: MAR ← (PC)	C_2
	t_2: MBR ← Memory PC ← (PC) + 1	C_5, C_R
	t_3: IR ← (MBR)	C_4
Indirect:	t_1: MAR ← (IR(Address))	C_8
	t_2: MBR ← Memory	C_5, C_R
	t_3: IR(Address) ← (MBR(Address))	C_4
Interrupt:	t_1: MBR ← (PC)	C_1
	t_2: MAR ← Save-address PC ← Routine-address	
	t_3: Memory ← (MBR)	C_{12}, C_W

C_R = Read control signal to system bus.
C_W = Write control signal to system bus.

Figure: Micro-operation and control signals.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.719. 154

Module 5c
Central Processing Unit (CPU)

5

Overview

- The term microprogram is an approach to Control Unit (CU) design that was organized systematically and avoided the complexities of a hardwired implementation.
- In recent years, microprogramming has become less used but remains a tool available to computer designers.
- **Example**: Pentium 4 → most of instructions are executed without the use of microprogramming, but some of the instructions are executed using microprogramming.

- A wide variety of techniques have been used for CU. Most of these fall into one of two categories:
  - (a) ……..
  - (b) ……..

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.730. 156

(a) Hardwired Implementation

5

- Sequential circuits.
- The control logic is implemented with gates, flip-flops, decoders, and other digital circuits.

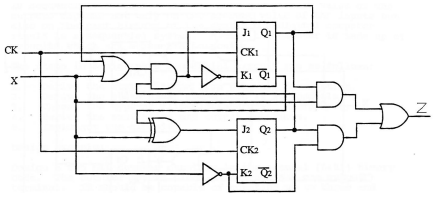


Figure: Example of sequential circuit.

Fast operation, small (requires less components).

Needs wiring change (if the design has to be modified)

Figure resource: https://web.sonoma.edu/users/m/marivani/es210/units/images/fig10_1.png (3 Apr 2019) 157

(b) Microprogrammed Implementation

5

- The control information is stored in a _____.
- the control memory is programmed to initiate the required sequence of micro-operations (μ**OP**).

Systematic, and any required change can be done by updating the micro-program in control memory.

Slow operation, requires more components.

158

5

Micro-Operations

- To implement a Control Unit (CU) as an interconnection of basic logic elements is no easy task.
- The design must include logic for sequencing through micro-operations, for executing micro-operations, for interpreting opcodes, and for making decisions based on ALU flags.

- It is difficult to design and test such a piece of hardware.
- The design is relatively inflexible. For example, it is difficult to change the design if one wishes to add a new machine instruction.

Disadvantages of hardware implementation.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.730. 159

5

Solution:

- An alternative to a hardwired control unit is a microprogrammed control unit, in which the logic of the CU is specified by a microprogram (_____).

- A microprogram consists of a sequence of instructions (*micro-operations*) in a microprogramming language.

- A microprogrammed control unit is a relatively simple logic circuit that is capable of :

  (1) sequencing through *micro-operations* and

  (2) generating control signals to execute each *micro-operation*.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.730. 160

How can we use the concept of microprogramming to implement a Control Unit (CU)?

5

- Consider that for each micro-operation, the CU is allowed to generate a set of control signals.
- Thus, for any micro-operation, each control line emanating from the CU is either *on* (*1*) or *off* (*0*).
- So we could construct a control word (_____) in which each bit represents one control line.

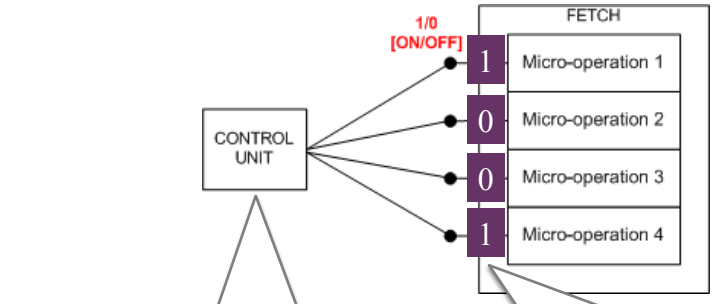
- Then each micro-operation would be represented by a different pattern of *1*s and *0*s in the microinstruction.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.732. 161

5

Example 20:

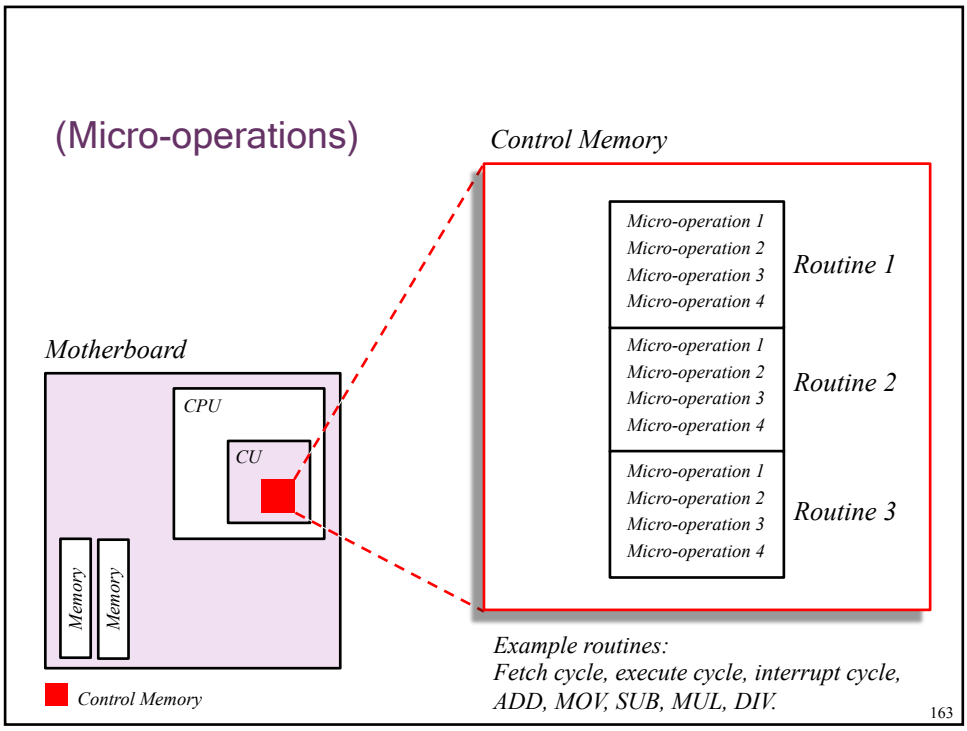
For *fetch instruction,* there are many micro-operations.

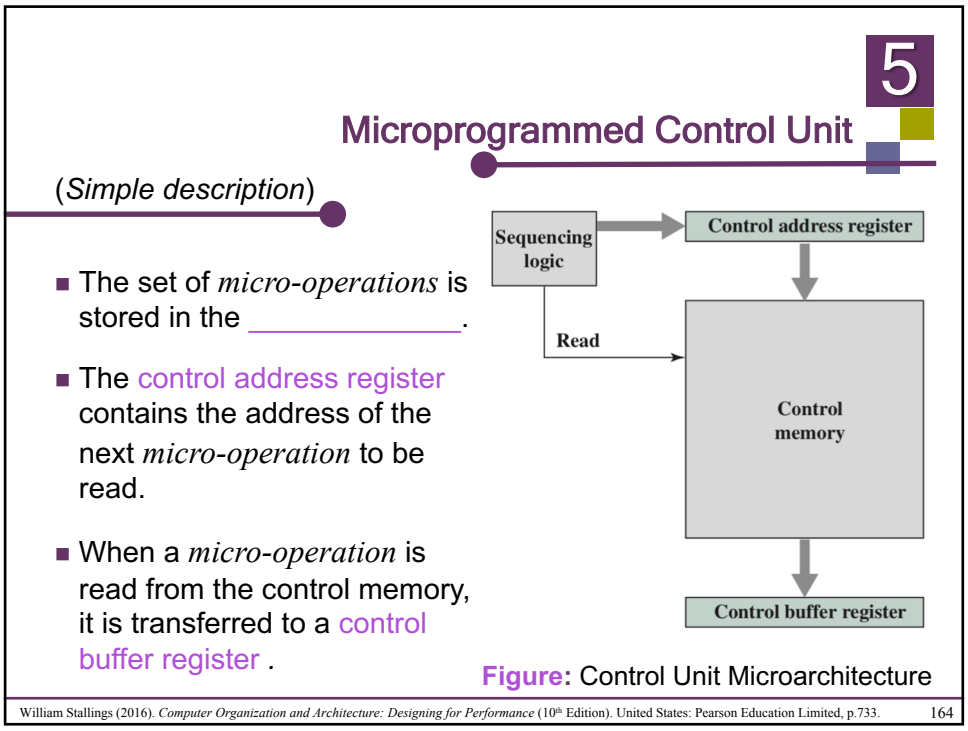


Control Unit sends on (1) or off (0) signal to the micro-operation

1010, 0011, 1100, 1110 will initiate the different micro-operation to do the fetch instruction.

162


(Micro-operations)
Control Memory
Micro-operation 1
Micro-operation 2
Micro-operation 3
Micro-operation 4
Routine 1
Micro-operation 1
Micro-operation 2
Micro-operation 3
Micro-operation 4
Routine 2
Micro-operation 1
Micro-operation 2
Micro-operation 3
Micro-operation 4
Routine 3
Motherboard
CPU
CU
Memory
Memory
Control Memory
Example routines:
Fetch cycle, execute cycle, interrupt cycle,
ADD, MOV, SUB, MUL, DIV.
163


5
Microprogrammed Control Unit
(Simple description)
The set of micro-operations is stored in the _____.
The control address register contains the address of the next micro-operation to be read.
When a micro-operation is read from the control memory, it is transferred to a control buffer register .
Sequencing logic
Control address register
Read
Control memory
Control buffer register
Figure: Control Unit Microarchitecture
William Stallings (2016). Computer Organization and Architecture: Designing for Performance (10th Edition). United States: Pearson Education Limited, p.733.
164

- Figure shows the programs (_____) could be arranged in a control memory.

The micro-operations in each routine are to be executed sequentially.

Each routine ends with a branch or jump instruction indicating where to go next.

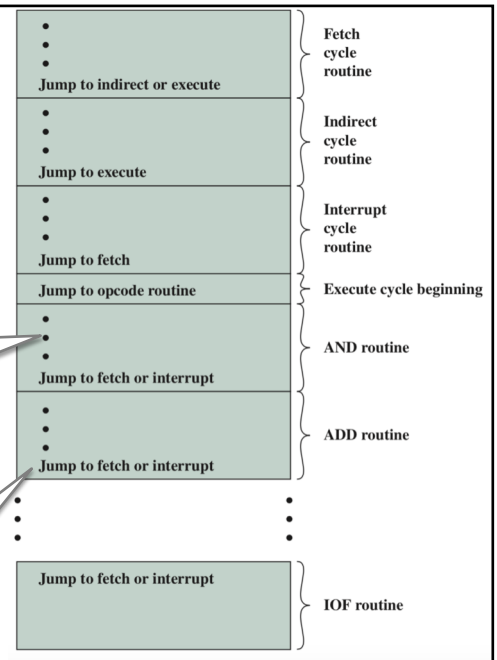


Figure: Organization of control memory

(*Detail description*)

- Comparing this figure with the previous figure (*simple description*), we see that the control unit still has the same inputs (IR, ALU flags, clock) and outputs (control signals).

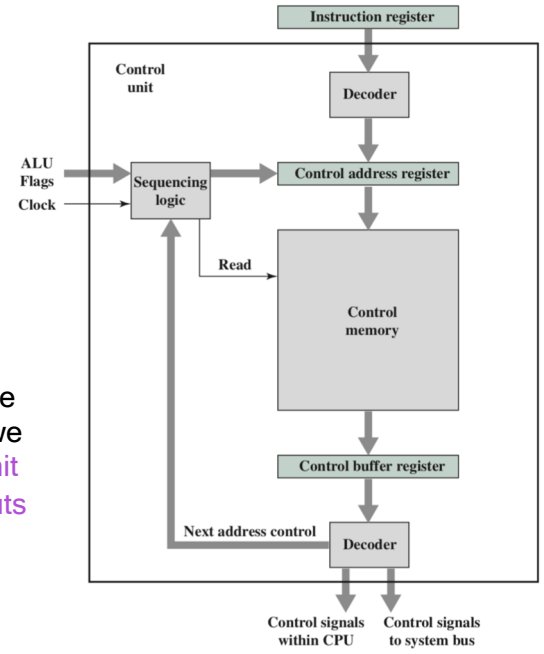
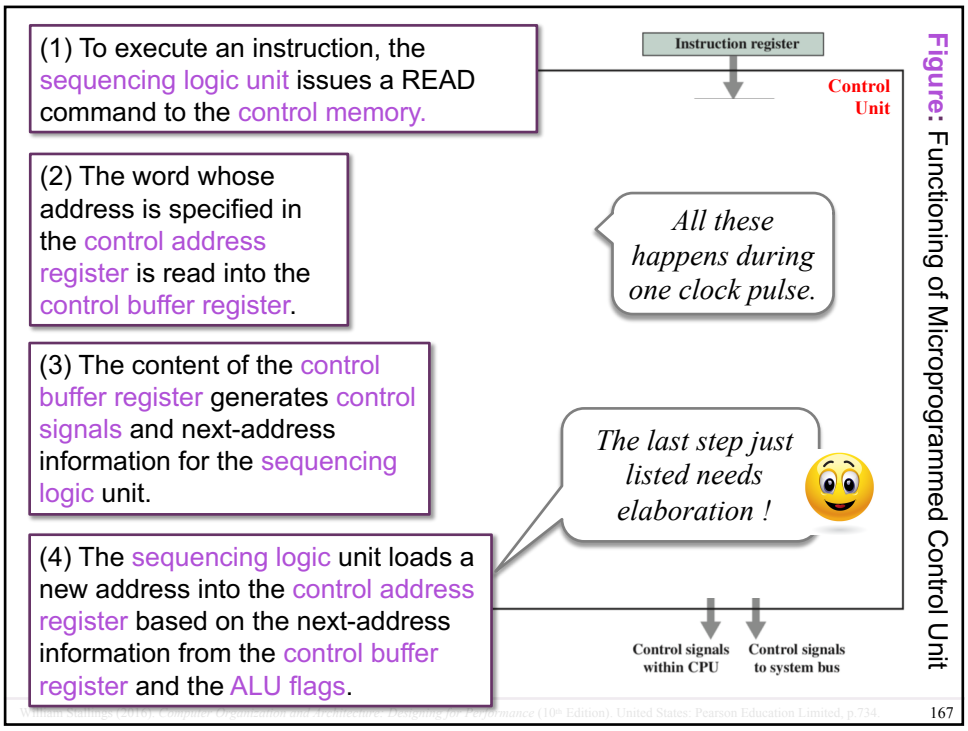
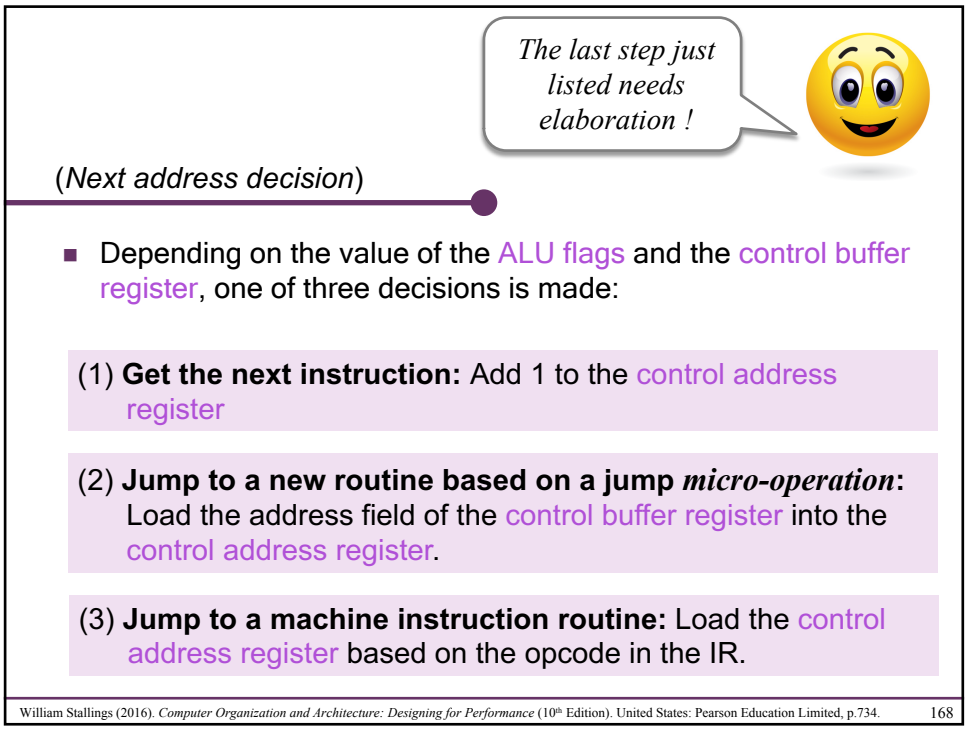


Figure: Functioning of Microprogrammed Control Unit


(1) To execute an instruction, the sequencing logic unit issues a READ command to the control memory.
(2) The word whose address is specified in the control address register is read into the control buffer register.
(3) The content of the control buffer register generates control signals and next-address information for the sequencing logic unit.
(4) The sequencing logic unit loads a new address into the control address register based on the next-address information from the control buffer register and the ALU flags.
Instruction register
Control Unit
All these happens during one clock pulse.
The last step just listed needs elaboration !
Control signals within CPU
Control signals to system bus
Figure: Functioning of Microprogrammed Control Unit
167


The last step just listed needs elaboration !
(Next address decision)
Depending on the value of the ALU flags and the control buffer register, one of three decisions is made:
(1) Get the next instruction: Add 1 to the control address register
(2) Jump to a new routine based on a jump micro-operation: Load the address field of the control buffer register into the control address register.
(3) Jump to a machine instruction routine: Load the control address register based on the opcode in the IR.
William Stallings (2016). Computer Organization and Architecture: Designing for Performance (10th Edition). United States: Pearson Education Limited, p.734.
168

5 Address Generation

- Another viewpoint is to consider the various ways in which the next address can be derived or computed.

Table: Microinstruction Address Generation Techniques

Two-field	Mapping
Unconditional branch	Addition
Conditional branch	Residual control

The address is explicitly available in the microinstruction.

Require additional logic to generate the address.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.743. 169

- *Microinstructions are stored in _____ in groups, which each group specifying a *routine*.
- *Each computer instruction has its own microprogram routine to generate *micro-operations* that execute the instruction.

***Mapping** The transformation from the instruction code bits (*opcode*) to an address in control memory where the routine (*micro-operation*) is located.

Mapping is one of several implicit techniques that commonly used !

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.743.
* Mano, Morris M. (1993). *Computer System Architecture* (3rd Edition). p.216. 170

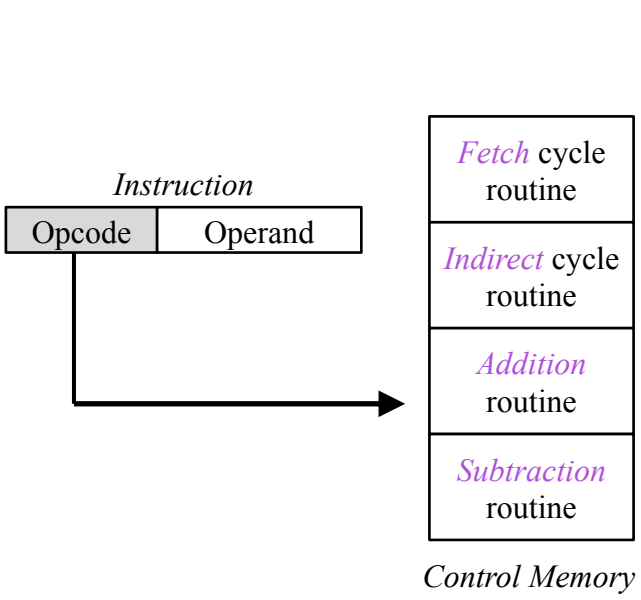


Figure: Transformation of opcode to microinstruction

171

ROM (Read-Only Memory)

Mapping of Instruction

- A special type of branch exist when a microinstruction specifies a branch to the first word in control memory where a microprogram routine for an instruction is located.
- Two methods:

Using ROM to specify mapping function

* Mano, Morris M. (1993). *Computer System Architecture* (3rd Edition). p.219.

172

5

(a) Direct Mapping

- The status bits for this type of branch are the bits in the operation code part of the instruction.

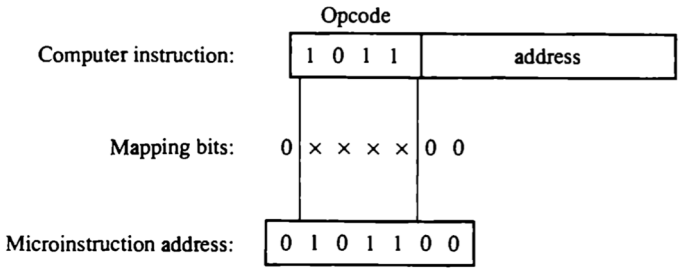


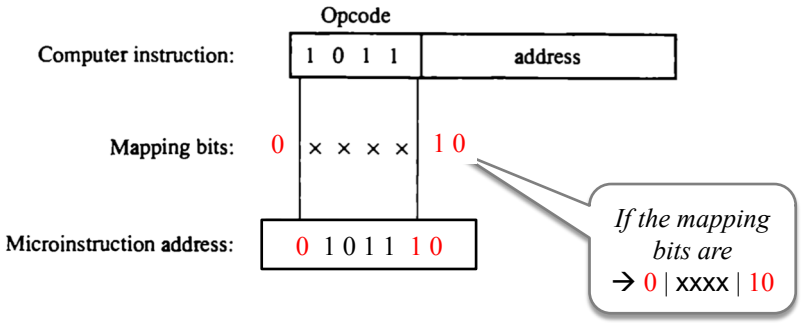
Figure: Mapping from instruction code to microinstruction address

* Mano, Morris M. (1993). *Computer System Architecture* (3rd Edition). p.219. 173

5

Example 21:

Mapping from instruction code to microinstruction address.



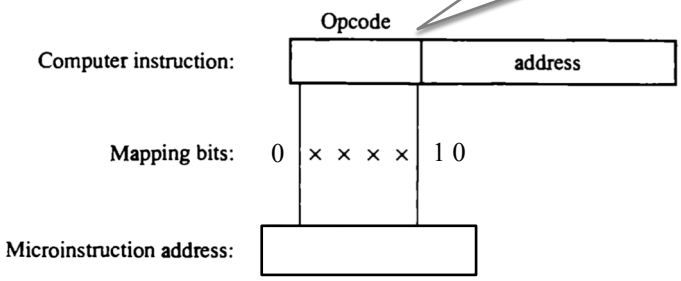
* Mano, Morris M. (1993). *Computer System Architecture* (3rd Edition). p.219. 174

5

Example 22:

Mapping from instruction code to microinstruction address.

If the opcode bits are → 1001



* Mano, Morris M. (1993). *Computer System Architecture* (3rd Edition). p.219.

175

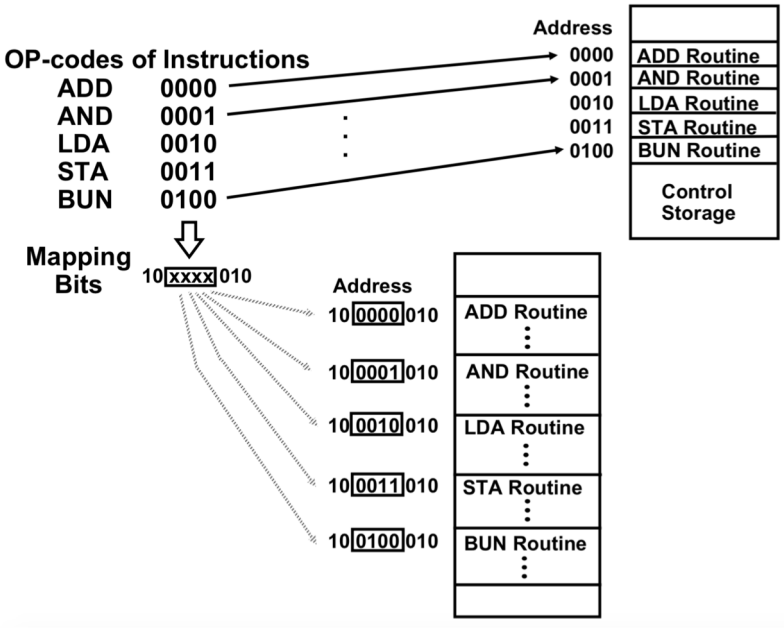


Figure: Direct mapping

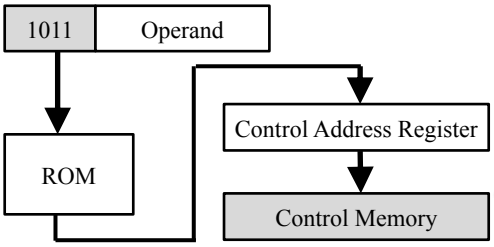
176

ROM (Read-Only Memory)

5

(b) Using ROM to Specify Mapping Function

- One can extend the concept of direct mapping with more general rules by using a ROM to specify the mapping function.
- The bits of the instruction specify the address of a mapping ROM.
- The contents of the mapping ROM gives the bits for the control address register.

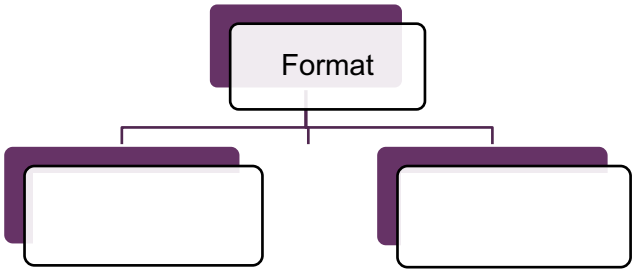


*The microprogram routine can be anywhere in the **control memory** → provides flexibility for adding instructions for **control memory** as need arise.*

* Mano, Morris M. (1993). *Computer System Architecture* (3rd Edition). p.219. 177

5

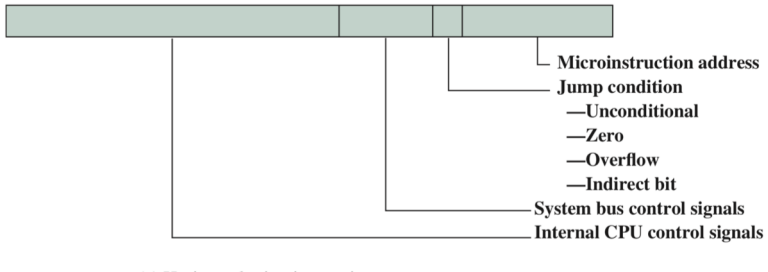
Microinstruction Format



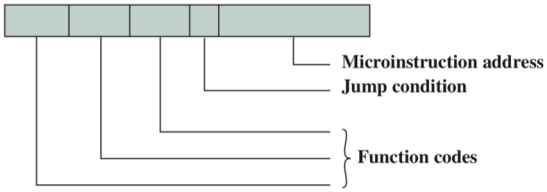
Each microinstruction specifies many different micro-operations to be performed in parallel.

Each microinstruction specifies single (or few) micro-operations to be performed.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.732. 178



(a) Horizontal microinstruction

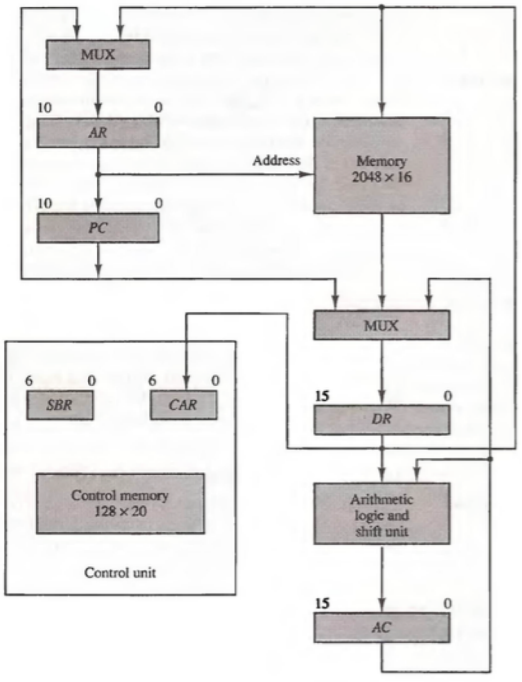


(b) Vertical microinstruction

Figure: Typical Microinstruction formats

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.732. 179

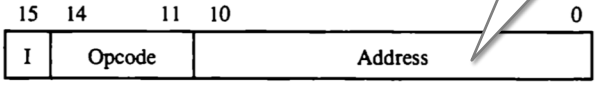
Figure: Computer hardware configuration



* Mano, Morris M. (1993). *Computer System Architecture* (3rd Edition). p.221. 180

I → Indirect Addressing

11 bits address field



(a) Instruction format

4 bits opcode can have 16 possible instructions

Symbol	OP-code	Description
ADD	0000	AC ← AC + M[EA]
BRANCH	0001	if (AC < 0) then (PC ← EA)
STORE	0010	M[EA] ← AC
EXCHANGE	0011	AC ← M[EA], M[EA] ← AC

EA (Effective Address)

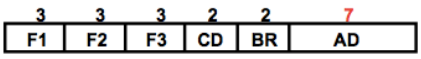
Figure: Sample of four computer instructions

* Mano, Morris M. (1993). *Computer System Architecture* (3rd Edition). p.222. 181

5

Vertical microinstruction format

Microinstruction Format



F1, F2, F3: Microoperation fields
CD: Condition for branching
BR: Branch field
AD: Address field

* Mano, Morris M. (1993). *Computer System Architecture* (3rd Edition). p.223. 182

F1	Micro-operation	Symbol
000	None	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLRAC
011	AC ← AC + 1	INCAC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

F2	Micro-operation	Symbol
000	None	NOP
001	AC ← AC - DR	SUB
010	AC ← AC ∨ DR	OR
011	AC ← AC ∧ DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR

F3	Micro-operation	Symbol
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← AC'	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	Reserved	

Microinstruction field description:
F1, F2, F3

* Mano, Morris M. (1993). *Computer System Architecture* (3rd Edition). p.224. 183

The condition for branching

Microinstruction field description:
CD, BR

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	DR(15)	I	Indirect address bit
10	AC(15)	S	Sign bit of AC
11	AC = 0	Z	Zero value in AC

The type of branch to be used

BR	Symbol	Function
00	JMP	CAR ← AD if condition = 1
		CAR ← CAR + 1 if condition = 0
01	CALL	CAR ← AD, SBR ← CAR + 1 if condition = 1
		CAR ← CAR + 1 if condition = 0
10	RET	CAR ← SBR (Return from subroutine)
11	MAP	CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0

* Mano, Morris M. (1993). *Computer System Architecture* (3rd Edition). p.224. 184

F2	Microoperation	Symbol
000	None	NOP
001	AC ← AC - DR	SUB
010	AC ← AC ∨ DR	OR
011	AC ← AC ∧ DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← AC'	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	Reserved	

Example 23:

A microinstruction with 2 simultaneous micro-operations.

DR ← M [AR] {F2 = ... }
PC ← PC + 1 {F3 = ... }

Since no F1 micro-operation, {F1 = ... }

Thus the vertical microinstruction fields:

F1	F2	F3	CD	BR	AD

* Mano, Morris M. (1993). *Computer System Architecture* (3rd Edition). p.223. 185

No more than 3 micro-operations can be chosen for a single microinstruction.

F1:
F2:
F3:

DR ← M [AR]
PC ← PC + 1

F2	Microoperation	Symbol
000	None	NOP
001	AC ← AC - DR	SUB
010	AC ← AC ∨ DR	OR
011	AC ← AC ∧ DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR

F1	Microoperation	Symbol
000	None	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLRAC
011	AC ← AC + 1	INCAC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

F3	Microoperation	Symbol
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← AC'	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	Reserved	

186

Partial Symbolic Microprogram

Label	Microops	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	

* Mano, Morris M. (1993). *Computer System Architecture* (3rd Edition). p.228. 187

Micro Routine	Address Decimal	Address Binary	Binary Microinstruction F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	000	01	01	1000011
	1	0000001	000	100	000	00	00	0000010
	2	0000010	001	000	000	00	00	1000000
	3	0000011	000	000	000	00	00	1000000
BRANCH	4	0000100	000	000	000	10	00	0000110
	5	0000101	000	000	000	00	00	1000000
	6	0000110	000	000	000	01	01	1000011
	7	0000111	000	000	110	00	00	1000000
STORE	8	0001000	000	000	000	01	01	1000011
	9	0001001	000	101	000	00	00	0001010
	10	0001010	111	000	000	00	00	1000000
	11	0001011	000	000	000	00	00	1000000
EXCHANGE	12	0001100	000	000	000	01	01	1000011
	13	0001101	001	000	000	00	00	0001110
	14	0001110	100	101	000	00	00	0001111
	15	0001111	111	000	000	00	00	1000000
FETCH	64	1000000	110	000	000	00	00	1000001
	65	1000001	000	100	101	00	00	1000010
	66	1000010	101	000	000	00	11	0000000
INDRCT	67	1000011	000	100	000	00	00	1000100
	68	1000100	101	000	000	00	10	0000000

This microprogram can be implemented using ROM

* Mano, Morris M. (1993). *Computer System Architecture* (3rd Edition). p.230. 188

5

Example 24: Calculation example.

A system uses a control memory of 1024 words of 32 bits each. The microinstruction has 3 fields and a 16-bit micro-operations.

a) How many bits are there in the branch address field and the selection field?

b) If there are 16 status in the system, how many bits of the branch logic are used to select a status bit?

c) How many bits are left to select an input for the multiplexers?

* Mano, Morris M. (1993). *Computer System Architecture* (3rd Edition). p.235. 189

5

Solution :

The keywords:

A system uses a control memory of 1024 words of 32 bits each. The microinstruction has 3 fields and a 16-bit micro-operations

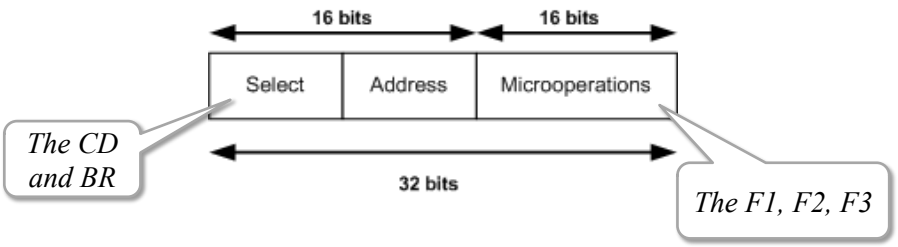


Figure: The microinstruction diagram

190

5

- The Control memory has 1024 words (2^{10}). Thus, the required address field for each word is 10 bits.
- The selection field consists of status bits and branch bits.
- The selection field will be: 32 – (16 + 10) = 6 bits

The micro-operations total bits

The bit for address field

191

5

a) How many bits are there in the branch address field and the selection field?

- The address field will be: 10 bit (2^{10} = 1024 words)
- The selection field will be: 32 – (16+10) = 6 bits

b) If there are 16 status in the system, how many bits of the branch logic are used to select a status bit?

- 16 status means 4 bits are required for it (2^4)

c) How many bits are left to select an input for the multiplexers?

- Input for the multiplexer = selection field bits – status bits
  = 6 – 4 = 2 bits

* Mano, Morris M. (1993). *Computer System Architecture* (3rd Edition). p.235.

192

CISC (Complex Instruction Set Computing)

5

Advantages & Disadvantages of Microprogramming

- It simplifies the design of the control unit.
- Thus, it is both cheaper and less error prone to implement.
- the decoders and sequencing logic unit are very simple pieces of logic.

- It will be somewhat slower than a hardwired unit of comparable technology

Despite this, microprogramming is the dominant technique for implementing CU in pure CISC architectures, due to its ease of implementation.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.737. 193

5

Summary 3

- Explained the concept of *micro-operations* and define the principal instruction cycle phases in terms of micro-operations.
- Discussed how micro-operations are organized to control a processor.
- Understand hardwired control unit organization.

William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.708. 194