USN | | | | | | | | | | |

IAT 3 – Nov. 2019

Sub:	Computer Organization					Sub Code:	18CS34	Branch:	CSE
Date:	18/11/2019	Duration:	90 mins	Max Marks:	50	Sem/Sec:	3 (A,B,C)		OBE

Answer **FOUR** FULL questions selecting AT LEAST ONE question **FROM EACH PART**

		MARKS	CO	RBT
1 (a)	**PART A** Write the sequence of control steps required for single bus structure for each if the following instructions: i) Add the contents of memory location whose address is at memory location NUM to register R1.	[10]	CO5	L3

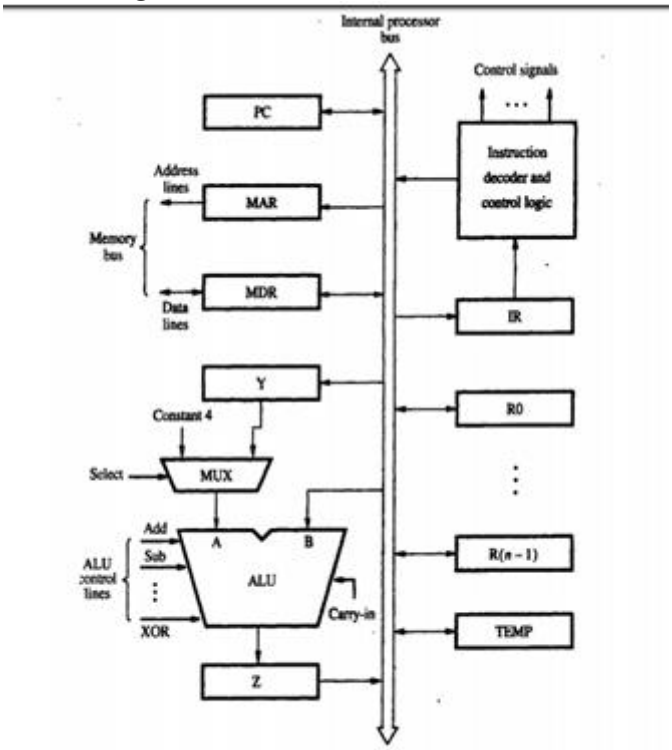


Figure 7.1 Single-bus organization of the datapath inside a processor.

ii)

1. PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2. Z_{out}, PC_{in}, Y_{in}, WMFC
3. MDR_{out}, IR_{in}
4. PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
5. Z_{out}, PC_{in}, WMFC
6. MDR_{out}, MAR_{in}, Read, WMFC
7. MDR_{out}, MAR_{in}, Read
8. $R1_{out}$, Y_{in}, WMFC
9. MDR_{out}, Add, Z_{in}
10. Z_{out}, $R1_{in}$, End

OR

2 (a)	Brief note on Rambus Memory	[4]	CO3	L2

- Rambus developed the implementation of narrow bus.
- Rambus technology is a fast signaling method used to transfer information betw
- The signals consist of much smaller voltage swings around a reference voltage \
- The reference voltage is about 2V.
- The two logical values are represented by 0.3V swings above and below V_{ref}.
- This type of signaling is generally is known as **Differential Signalling**.
- Rambus provides a complete specification for design of communication called as
- Rambus memory has a clock frequency of 400 MHz.
- The data are transmitted on both the edges of clock so that effective data-trans
- Circuitry needed to interface to Rambus channel is included on chip. Such chips (RDRAM = Rambus DRAMs).
- Rambus channel has:
  1) 9 Data-lines (1^{st}-8^{th} line ->Transfer the data, 9^{th} line->Parity checking).
  2) Control-Line &
  3) Power line.
- A two channel rambus has 18 data-lines which has no separate Address-Lines.
- Communication between processor and RDRAM modules is carried out b transmitted on the data-lines.
- There are 3 types of packets:
  1) Request
  2) Acknowledge &
  3) Data.

(b) Explain Memory Hierarchy with respect to cost, speed and size. [6]

CO3 L2

, SIZE, and COST

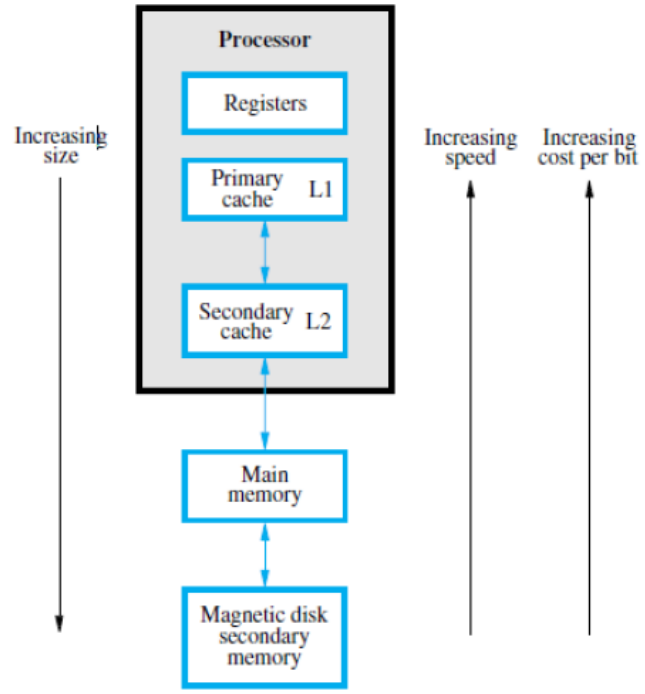


Figure 8.14 Memory hierarchy.

- Fastest access is to the data held in processor registers. Registers are at the top of the memory hierarchy.
- Relatively small amount of memory that can be implemented on the processor chip. This is processor cache. Usually implemented as SRAM.
- Two levels of cache.

Level 1 (L1) cache is on the processor chip.
Level 2 (L2) cache is in between main memory and processor.

- Next level is main memory, implemented as DRAM (SIMMs,RIMM,DIMM). Much larger, but much slower than cache memory.
- Next level is magnetic disks. Huge amount of inexepensive storage.
- Speed of memory access is critical, the idea is to bring instructions and data that will be used in the near future as close to the processor as possible.

PART B

3 (a)	With a figure, explain single bus organization of Datapath inside a processor.	[10]	CO5	L2

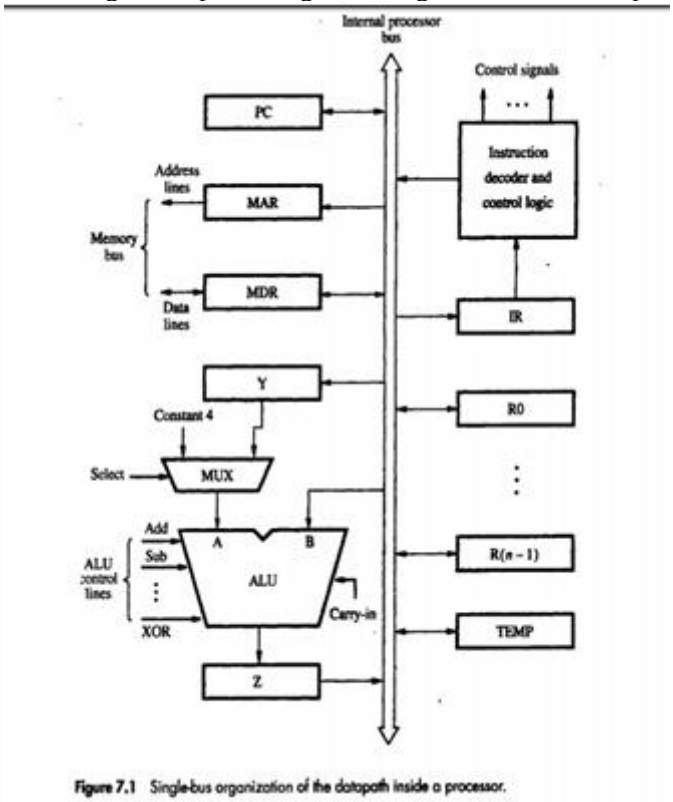


Figure 7.1 Single-bus organization of the datapath inside a processor.

SINGLE BUS ORGANIZATION

- ALU and all the registers are interconnected via a **Single Common Bus** (Figure 7.1).
- Data & address lines of the external memory-bus is connected to the internal processor-bus via MDR & MAR respectively. (MDR→ Memory Data Register, MAR → Memory Address Register).
- **MDR** has 2 inputs and 2 outputs. Data may be loaded
  - → into MDR either from memory-bus (external) or
  - → from processor-bus (internal).
- **MAR**'s input is connected to internal-bus;
  MAR's output is connected to external-bus.
- **Instruction Decoder & Control Unit** is responsible for
  - → issuing the control-signals to all the units inside the processor.
  - → implementing the actions specified by the instruction (loaded in the IR).
- Register R0 through R(n-1) are the **Processor Registers**.
  The programmer can access these registers for general-purpose use.
- Only processor can access 3 registers **Y**, **Z** & **Temp** for temporary storage during program-execution.
  The programmer cannot access these 3 registers.
- In **ALU**, 1) 'A' input gets the operand from the output of the multiplexer (MUX).
  2) 'B' input gets the operand directly from the processor-bus.
- There are 2 options provided for 'A' input of the ALU.
- MUX is used to select one of the 2 inputs.
- **MUX** selects either
  - → output of Y or
  - → constant-value 4(which is used to increment PC content).

- An instruction is executed by performing one or more of the following operations:
  1) Transfer a word of data from one register to another or to the ALU.
  2) Perform arithmetic or a logic operation and store the result in a register.
  3) Fetch the contents of a given memory-location and load them into a register.
  4) Store a word of data from a register into a given memory-location.
- **Disadvantage:** Only one data-word can be transferred over the bus in a clock cycle.
  Solution: Provide multiple internal-paths. Multiple paths allow several data-transfers to take place in parallel.

OR

4 (a) Explain the differences between Hardwired and Micro-programmed control with neat diagram. [10] CO5 L2

Attribute	Hardwired Control	Microprogrammed Control
Definition	Hardwired control is a control mechanism to generate control-signals by using gates, flip-flops, decoders, and other digital circuits.	Micro programmed control is a control mechanism to generate control-signals by using a memory called control store (CS), which contains the control-signals.
Speed	Fast	Slow
Control functions	Implemented in hardware.	Implemented in software.
Flexibility	Not flexible to accommodate new system specifications or new instructions.	More flexible, to accommodate new system specification or new instructions redesign is required.
Ability to handle large or complex instruction sets	Difficult.	Easier.
Ability to support operating systems & diagnostic features	Very difficult.	Easy.
Design process	Complicated.	Orderly and systematic.
Applications	Mostly RISC microprocessors.	Mainframes, some microprocessors.
Instructionset size	Usually under 100 instructions.	Usually over 100 instructions.
ROM size	-	2K to 10K by 20-400 bit microinstructions.
Chip area efficiency	Uses least area.	Uses more area.

PART C

5 (a) What is cache memory. Explain the following terms a) write through b) write back c) early restart d) Miss penalty e) average memory access time f) dirty bit g) valid bit h) Hit rate i) Write buffer [10] | CO3 | L2

ıe effectiveness of cache mechanism is based on the property of **'Locality of Reference'.**
ality of Reference
any instructions in the localized areas of program are executed repeatedly during some time p
emainder of the program is accessed relatively infrequently (Figure 8.15).
ıere are 2 types:
) Temporal
· The recently executed instructions are likely to be executed again very soon.
) Spatial
· Instructions in close proximity to recently executed instruction are also likely to be executed
active segment of program is placed in cache-memory, then total execution time can be redu
ock refers to the set of contiguous address locations of some size.
ıe cache-line is used to refer to the cache-block.

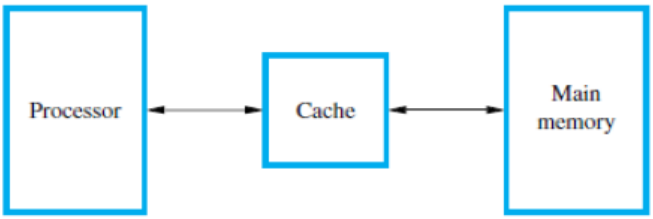


Figure 8.15 Use of a cache memory.

e Cache-memory stores a reasonable number of blocks at a given time.
is number of blocks is small compared to the total number of blocks available in main
ırrespondence b/w main-memory-block & cache-memory-block is specified by mappin
ıche control hardware decides which block should be removed to create space for the
e collection of rule for making this decision is called the **Replacement Algorithm.**
e cache control-circuit determines whether the requested-word currently exists in the

HIT

- If the data is in the cache it is called a **Read or Write hit**.
- Read hit:
  - The data is obtained from the cache.
- Write hit:
  - Cache has a replica of the contents of the main memory.
  - Contents of the cache and the main memory may be updated simultaneously. This is the write-through protocol.
  - Update the contents of the cache, and mark it as updated by setting a bit known as the dirty bit or modified bit. The contents of the main memory are updated when this block is replaced. This is write-back or copy-back protocol.

MISS

When the addressed word in a Read operation is not in the cache, a *read miss* occurs. The block of words that contains the requested word is copied from the main memory into the cache. After the entire block is loaded into the cache, the particular word requested is forwarded to the processor. Alternatively, this word may be sent to the processor as soon as it is read from the main memory. The latter approach, which is called *load-through,* or *early restart,* reduces the processor's waiting period somewhat, but at the expense of more complex circuitry.

During a Write operation, if the addressed word is not in the cache, a *write miss* occurs. Then, if the write-through protocol is used, the information is written directly into the main memory. In the case of the write-back protocol, the block containing the addressed word is first brought into the cache, and then the desired word in the cache is overwritten with the new information.

MAPPING FUNCTION

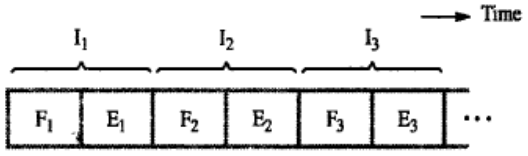
OR

6(a) Explain basic concepts of pipeling in details . Also explain the hazards [10]

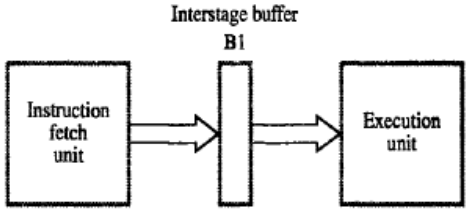
PIPELING

BASIC CONCEPTS

The speed of execution of programs is influenced by many factors. One way to improve performance is to use faster circuit technology to build the processor and the main memory. Another possibility is to arrange the hardware so that more than one operation can be performed at the same time. In this way, the number of operations performed per second is increased even though the elapsed time needed to perform any one operation is not changed.

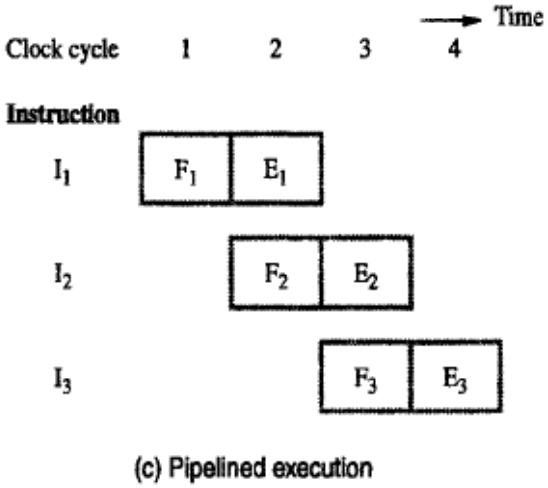


(a) Sequential execution



(b) Hardware organization

CO5 L2



(c) Pipelined execution

Figure 8.1 Basic idea of instruction pipelining.

The computer is controlled by a clock whose period is such that the fetch and execute steps of any instruction can each be completed in one clock cycle. Operation of the computer proceeds as in Figure 8.1*c*. In the first clock cycle, the fetch unit fetches an instruction I_1 (step F_1) and stores it in buffer B1 at the end of the clock cycle. In the second clock cycle, the instruction fetch unit proceeds with the fetch operation for instruction I_2 (step F_2). Meanwhile, the execution unit performs the operation specified by instruction I_1, which is available to it in buffer B1 (step E_1). By the end of the second clock cycle, the execution of instruction I_1 is completed and instruction I_2 is available. Instruction I_2 is stored in B1, replacing I_1, which is no longer needed. Step E_2 is performed by the execution unit during the third clock cycle, while instruction I_3 is being fetched by the fetch unit. In this manner, both the fetch and execute units are kept busy all the time. If the pattern in Figure 8.1*c* can be sustained for a long time, the completion rate of instruction execution will be twice that achievable by the sequential operation depicted in Figure 8.1*a*.

In summary, the fetch and execute units in Figure 8.1*b* constitute a two-stage pipeline in which each stage performs one step in processing an instruction. An inter-stage storage buffer, B1, is needed to hold the information being passed from one stage to the next. New information is loaded into this buffer at the end of each clock cycle.

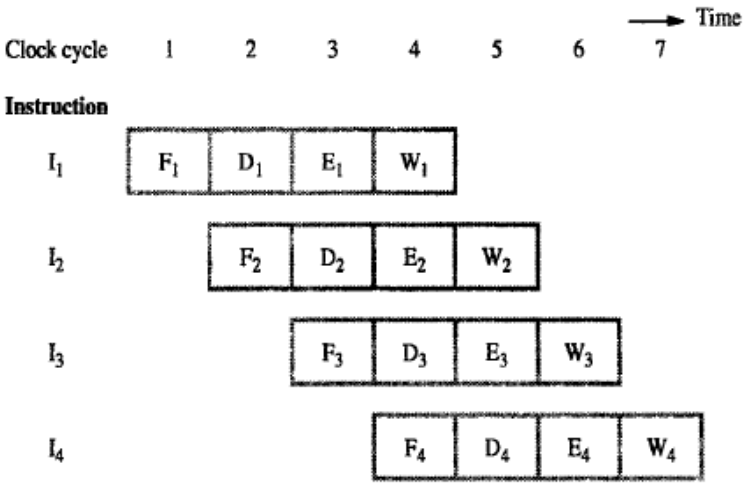
The processing of an instruction need not be divided into only two steps. For example, a pipelined processor may process each instruction in four steps, as follows:

- F Fetch: read the instruction from the memory.
- D Decode: decode the instruction and fetch the source operand(s).
- E Execute: perform the operation specified by the instruction.
- W Write: store the result in the destination location.

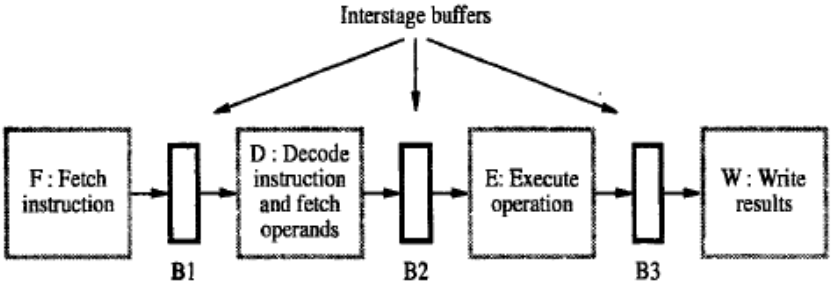
ROLE OF CACHE MEMORY

Each stage in a pipeline is expected to complete its operation in one clock cycle. Hence, the clock period should be sufficiently long to complete the task being performed in any stage. If different units require different amounts of time, the clock period must allow the longest task to be completed. A unit that completes its task early is idle for the remainder of the clock period. Hence, pipelining is most effective in improving performance if the tasks being performed in different stages require about the same amount of time.

This consideration is particularly important for the instruction fetch step, which is assigned one clock period in Figure 8.2*a*. The clock cycle has to be equal to or greater than the time needed to complete a fetch operation. However, the access time of the main memory may be as much as ten times greater than the time needed to perform basic pipeline stage operations inside the processor, such as adding two numbers. Thus, if each instruction fetch required access to the main memory, pipelining would be of little value.



(a) Instruction execution divided into four steps



(b) Hardware organization

PIPELINE PERFORMANCE

For a variety of reasons, one of the pipeline stages may not be able to complete its processing task for a given instruction in the time allotted. For example, stage E in the four-stage pipeline of Figure 8.2*b* is responsible for arithmetic and logic operations, and one clock cycle is assigned for this task. Although this may be sufficient for most operations, some operations, such as divide, may require more time to complete. Figure 8.3 shows an example in which the operation specified in instruction I_2 requires three cycles to complete, from cycle 4 through cycle 6. Thus, in cycles 5 and 6, the Write stage must be told to do nothing, because it has no data to work with. Meanwhile, the information in buffer B2 must remain intact until the Execute stage has completed its operation. This means that stage 2 and, in turn, stage 1 are blocked from accepting new instructions because the information in B1 cannot be overwritten. Thus, steps D_4 and F_5 must be postponed as shown.

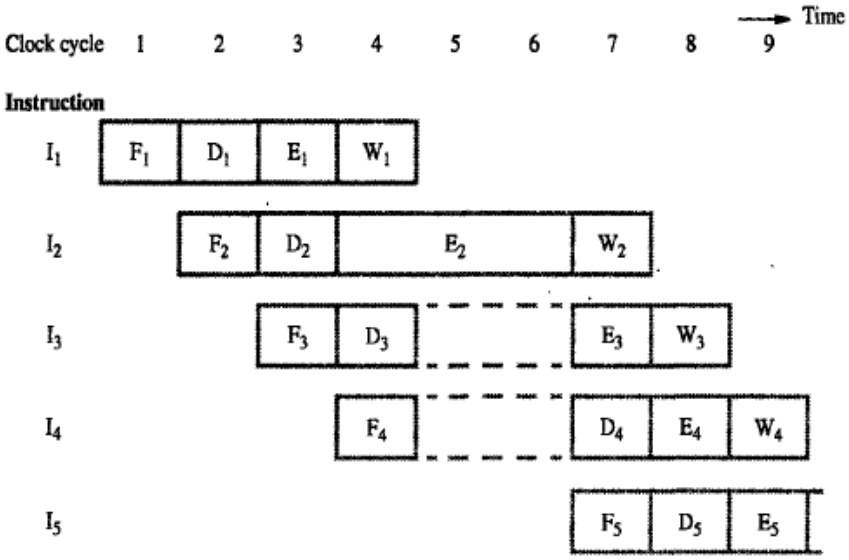
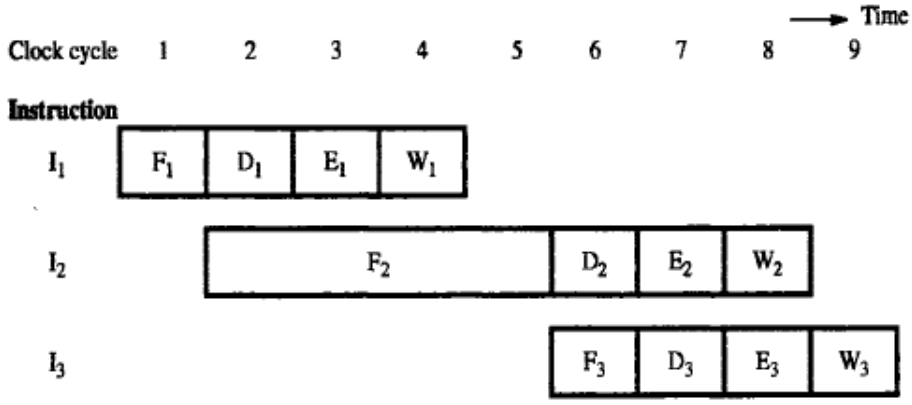


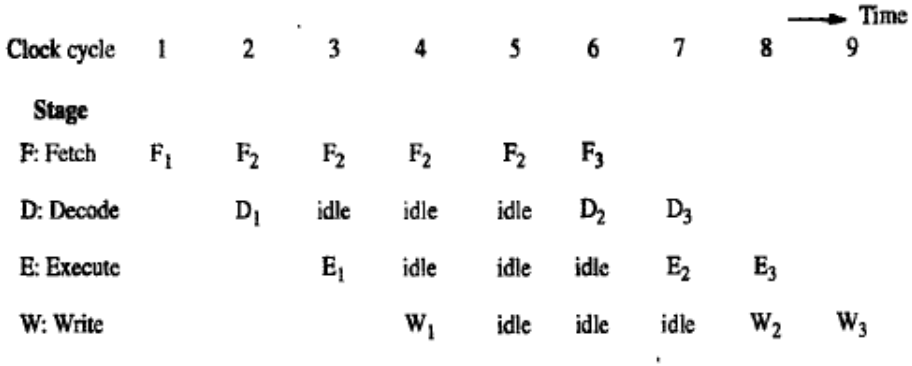
Figure 8.3 Effect of an execution operation taking more than one clock cycle.

Pipelined operation in Figure 8.3 is said to have been *stalled* for two clock cycles. Normal pipelined operation resumes in cycle 7. Any condition that causes the pipeline to stall is called a *hazard*. We have just seen an example of a *data hazard*. A data hazard is any condition in which either the source or the destination operands of an instruction are not available at the time expected in the pipeline. As a result some operation has to be delayed, and the pipeline stalls.

The pipeline may also be stalled because of a delay in the availability of an instruction. For example, this may be a result of a miss in the cache, requiring the instruction to be fetched from the main memory. Such hazards are often called *control hazards* or *instruction hazards*. The effect of a cache miss on pipelined operation is illustrated in Figure 8.4. Instruction I_1 is fetched from the cache in cycle 1, and its execution proceeds normally. However, the fetch operation for instruction I_2, which is started in cycle 2, results in a cache miss. The instruction fetch unit must now suspend any further fetch requests and wait for I_2 to arrive. We assume that instruction I_2 is received and loaded into buffer B1 at the end of cycle 5. The pipeline resumes its normal operation at that point.



(a) Instruction execution steps in successive clock cycles



(b) Function performed by each processor stage in successive clock cycles

Figure 8.4 Pipeline stall caused by a cache miss in F2.

PART D

7(a) Explain three types of mapping functions for cache memory. [15] CO3 L2

(Direct mapping-5M, Associative Mapping- 5M, Set-associative mapping- 5M)

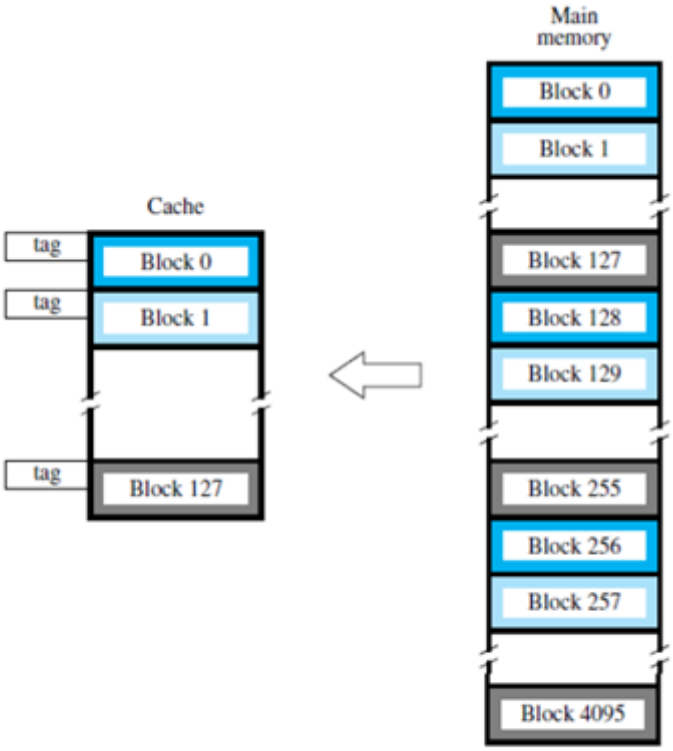


Figure 8.16 Direct-mapped cache.

DIRECT MAPPING

This technique is easy to implement but not very flexible.

Block j of the main memory maps onto j modulo 128 of the cache. For example, whenever **one of** the main memory blocks 0, 128, 256, ... Is loaded in the cache, it is stored in cache block 0 . Main memory blocks 1, 129, 257, ... are stored in cache block 1 (one at a time), and so on. Contention may occur for a single cache block required by multiple memory blocks. E.g when for program execution both memory block 1 and 129 are required but cache block 1 can only store one memory block. To resolve this, new blocks are allowed to overwrite the currently resident block.

From example,

4096 memory blocks need to be mapped to 128 cache blocks. i.e, each cache block identified 32 memory blocks(4096/128).

Main memory address is divided into three parts:

Tag (5 bits): identify which memory block (out of 32 in this case) is currently resident in the cache

Block (7 bits): cache block position where the new memory block must be stored

Word (4 bits): selects one of the words of the memory block (out of 16 words per block in this case)

ASSOCIATIVE MAPPING

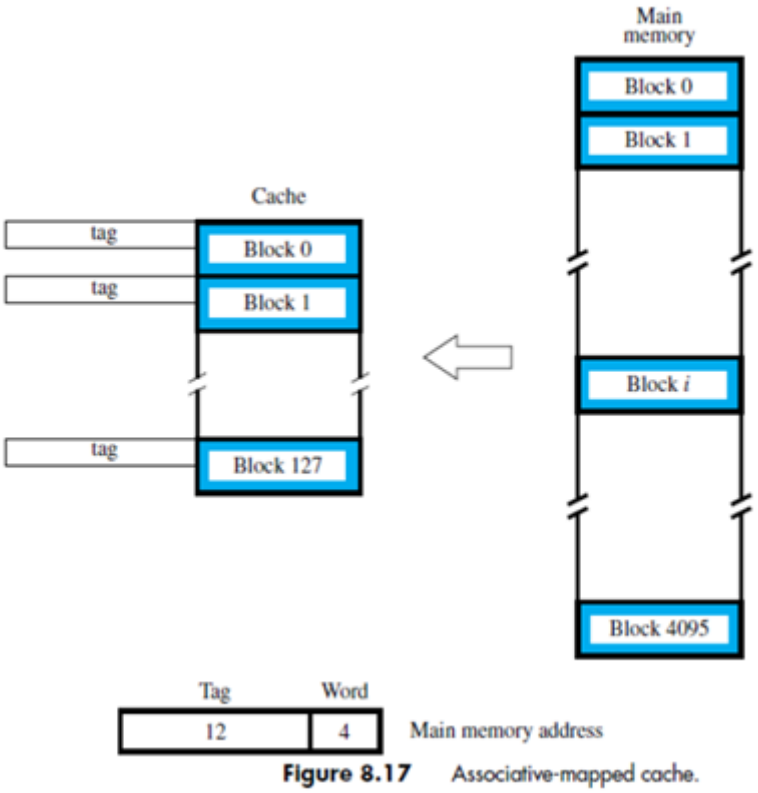


Figure 8.17 Associative-mapped cache.

- It is more flexible than direct mapping technique but more expensive. Main memory block can be placed into any cache block position.
- Memory address is divided into two fields:
  - Low order 4 bits identify the memory word within a block.
  - High order 12 bits or tag bits identify a memory block when residing in the cache.
- Flexible, and uses cache space efficiently.
- Replacement algorithms can be used to replace an existing block in the cache when the cache is full.
- Cost is higher than direct-mapped cache because of the need to search all 128 patterns to determine whether a given block is in the cache.

SET-ASSOCIATIVE MAPPING

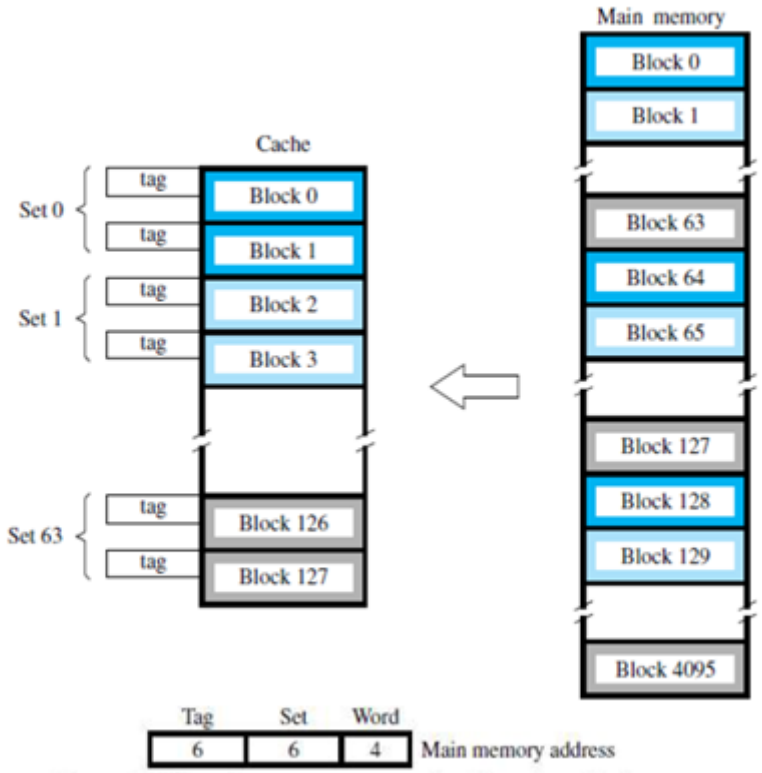


Figure 8.18 Set-associative-mapped cache with two blocks per set.

It is a **combination** of direct mapping and associative mapping techniques.
Blocks of the cache are grouped into **sets**, and the mapping allows a block of the main memory to reside in **any block of a specific set**.
Contention problem of direct mapping is eased by having a few choices for block placement. Hardware cost is reduced by decreasing the associative search.

(b) Briefly explain replacement algorithms for cache memory. [5] | CO3 | L2

Example:
Consider cache controller is tracking a set of four blocks in a set associative cache.
A 2-bit counter is used for each block (00=0, 01=1, 10=2, 11=3)

a) Hit occurs:
The counter of the block that is referenced is set to 0. Other block counters having value less than the value of the referenced counter are incremented by 1. Block counters having value greater than the value of referenced counter are remain unchanged.
Initially: 2, 3, 0, 1
Hit occurs for 2
Finally: 0 (after set to 0), 3 (unchanged), 1 (after increment), 2 (after increment)

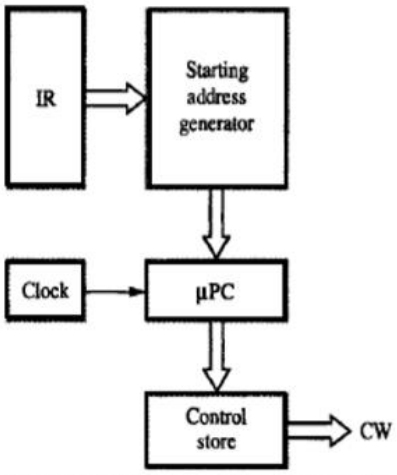
b) Miss occurs (Set not full):
The counter of the block where new block is loaded from memory is set to 0. All other block counters value is incremented by 1.
Initially: 2, 1, 0, _
Miss occurs
Finally: 3, 2, 1, **0** (new)

c) Miss occurs (Set full):
The block with the highest counter value i.e, 3 is removed. The new block is put into its place and its counter value is set to 0. All other three blocks counter values are incremented by 1.
Initially: 2, 3, 0, 1
Miss occurs
Finally: 3, 0 (replaced), 1, 2

OR

8 (a) ain with a neat diagram microprogrammed control. Also explain branching in [12]
oprogrammed control

roprogrammed Control



e 7.16 Basic organization of a microprogrammed control unit.

roprogramming is a method of control unit design (Figure 7.16).
ntrol-signals are generated by a program similar to machine language programs.
ntrol Word(CW) is a word whose individual bits represent various control-signals (like Add, PC
:h of the control-steps in control sequence of an instruction defines a unique combination of 1
CW.
ividual control-words in microroutine are referred to as **microinstructions** (Figure 7.15).
sequence of CWs corresponding to control-sequence of a machine instruction constitutes
oroutine.
e microroutines for all instructions in the instruction-set of a computer are stored in a spe
ory called the **Control Store (CS)**.
ontrol-unit generates control-signals for any instruction by sequentially reading CWs
sponding microroutine from CS.
C is used to read CWs sequentially from CS. (µPC→ Microprogram Counter).
ery time new instruction is loaded into IR, o/p of **Starting Address Generator** is loaded into µI
en, µPC is automatically incremented by clock;
causing successive microinstructions to be read from CS.
Hence, control-signals are delivered to various parts of processor in correct sequence.

antages
simplifies the design of control unit. Thus it is both, cheaper and less error prone implement.
ntrol functions are implemented in software rather than hardware.
e design process is orderly and systematic.
ore flexible, can be changed to accommodate new system specifications or to correct the de
rs quickly and cheaply.
mplex function such as floating point arithmetic can be realized efficiently.
dvantages
nicroprogrammed control unit is somewhat slower than the hardwired control unit, because tim
ired to access the microinstructions from CM.
e flexibility is achieved at some extra hardware cost due to the control memory and its acc
itry.

p	Action
	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
	Z_{out}, PC_{in}, Y_{in}, WMFC
	MDR_{out}, IR_{in}
	$R3_{out}$, MAR_{in}, Read
	$R1_{out}$, Y_{in}, WMFC
	MDR_{out}, SelectY, Add, Z_{in}
	Z_{out}, $R1_{in}$, End

e 7.6 Control sequence for execution of the instruction Add (R3),R1

Microinstructions

CO5 L2

ro - ruction	..	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	WMFC	End	:
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	
5		0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	
6		0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	
7		0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	

ure 7.15 An example of microinstructions for Figure 7.6.

ORGANIZATION OF MICROPROGRAMMED CONTROL UNIT TO SUPPORT CONDI BRANCHING

- **Drawback of previous Microprogram control:**
  - ➢ It cannot handle the situation when the control unit is required to check the status of condition codes or external inputs to choose between alternative courses of action.

Solution:

- ➢ Use conditional branch microinstruction.

- In case of conditional branching, microinstructions specify which of the external inputs, c codes should be checked as a condition for branching to take place.
- **Starting and Branch Address Generator Block** loads a new address into μPC microinstruction instructs it to do so (Figure 7.18).
- To allow implementation of a conditional branch, inputs to this block consist of
  - → external inputs and condition-codes &
  - → contents of IR.
- μPC is incremented every time a new microinstruction is fetched from microprogram memor in following situations:
  - 1) When a new instruction is loaded into IR, μPC is loaded with starting-address of mic for that instruction.
  - 2) When a Branch microinstruction is encountered and branch condition is satisfied loaded with branch-address.
  - 3) When an End microinstruction is encountered, μPC is loaded with address of firs microroutine for instruction fetch cycle.

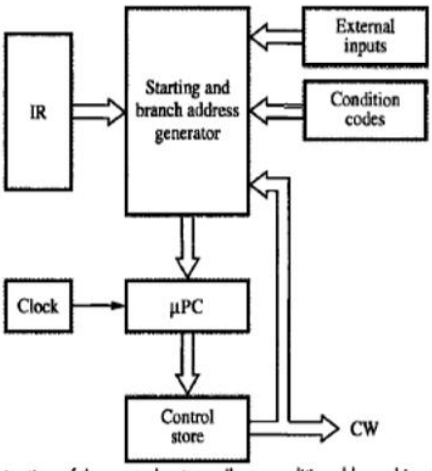


Figure 7.18 Organization of the control unit to allow conditional branching in the microprogram.

Address	Microinstruction
0	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
1	Z_{out}, PC_{in}, Y_{in}, WMFC
2	MDR_{out}, IR_{in}
3	Branch to starting address of appropriate microroutine
25	If N=0, then branch to microinstruction 0
26	Offset-field-of-IR_{out}, SelectY, Add, Z_{in}
27	Z_{out}, PC_{in}, End

Figure 7.17 Microroutine for the instruction Branch < 0.

MICROINSTRUCTIONS

• A simple way to structure microinstructions is to assign one bit position to each con required in the CPU.

• There are 42 signals and hence each microinstruction will have 42 bits.

• **Drawbacks of microprogrammed control:**

1) Assigning individual bits to each control-signal results in long microinstructions bec the number of required signals is usually large.
2) Available bit-space is poorly used because only a few bits are set to 1 in any given microinstruction.

• **Solution:** Signals can be grouped because

1) Most signals are not needed simultaneously.
2) Many signals are mutually exclusive. E.g. only 1 function of ALU can be activated at

For ex: Gating signals: IN and OUT signals (Figure 7.19).
Control-signals: Read, Write.
ALU signals: Add, Sub, Mul, Div, Mod.

• Grouping control-signals into fields requires a little more hardware because decoding-circuits must be used to decode bit patterns of each field into individual contro

• **Advantage:** This method results in a smaller control-store (only 20 bits are needed to patterns for the 42 signals).

MICROINSTRUCTIONS

• A simple way to structure microinstructions is to assign one bit position to each cont required in the CPU.

• There are 42 signals and hence each microinstruction will have 42 bits.

• **Drawbacks of microprogrammed control:**

1) Assigning individual bits to each control-signal results in long microinstructions beca the number of required signals is usually large.
2) Available bit-space is poorly used because only a few bits are set to 1 in any given microinstruction.

• **Solution:** Signals can be grouped because

1) Most signals are not needed simultaneously.
2) Many signals are mutually exclusive. E.g. only 1 function of ALU can be activated at

For ex: Gating signals: IN and OUT signals (Figure 7.19).
Control-signals: Read, Write.
ALU signals: Add, Sub, Mul, Div, Mod.

• Grouping control-signals into fields requires a little more hardware because decoding-circuits must be used to decode bit patterns of each field into individual control·

• **Advantage:** This method results in a smaller control-store (only 20 bits are needed to patterns for the 42 signals).

Microinstruction

F1	F2	F3	F4	F5

F1 (4 bits)	F2 (3 bits)	F3 (3 bits)	F4 (4 bits)	F5 (2 bits)
0000: No transfer 0001: PC_{out} 0010: MDR_{out} 0011: Z_{out} 0100: $R0_{out}$ 0101: $R1_{out}$ 0110: $R2_{out}$ 0111: $R3_{out}$ 1010: $TEMP_{out}$ 1011: $Offset_{out}$	000: No transfer 001: PC_{in} 010: IR_{in} 011: Z_{in} 100: $R0_{in}$ 101: $R1_{in}$ 110: $R2_{in}$ 111: $R3_{in}$	000: No transfer 001: MAR_{in} 010: MDR_{in} 011: $TEMP_{in}$ 100: Y_{in}	0000: Add 0001: Sub ⋮ 1111: XOR 16 ALU functions	00: No action 01: Read 10: Write

F6	F7	F8	...

F6 (1 bit)	F7 (1 bit)	F8 (1 bit)
0: SelectY 1: Select4	0: No action 1: WMFC	0: Continue 1: End

Figure 7.19 An example of a partial format for field-encoded microinstructions.

(b) Explain with a neat diagram hardwired control. [8] CO5 L2

Hardwired Control

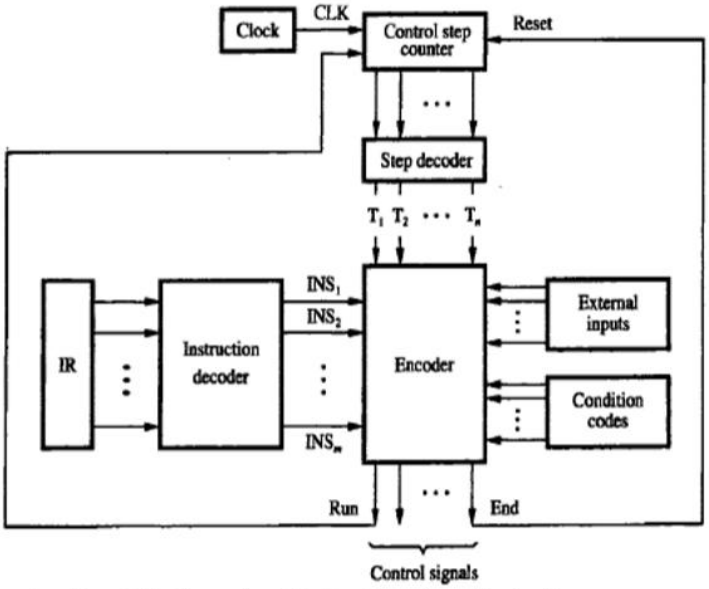


Figure 7.11 Separation of the decoding and encoding functions.

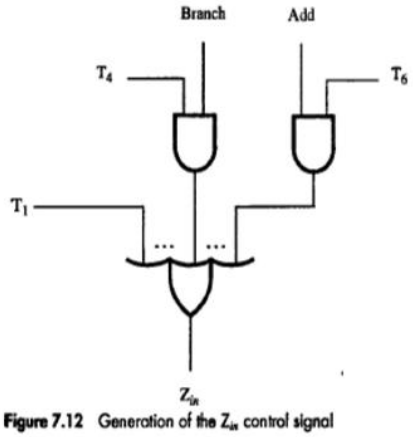


Figure 7.12 Generation of the Z_{in} control signal

- Hardwired control is a method of control unit design (Figure 7.11).
- The control-signals are generated by using logic circuits such as gates, flip-flops, decoders etc.
- **Decoder/Encoder Block** is a combinational-circuit that generates required control-outputs depending on state of all its inputs.
- **Instruction Decoder**
  - It decodes the instruction loaded in the IR.
  - If IR is an 8 bit register, then instruction decoder generates 2^8(256 lines); one for each instruction.
  - It consists of a separate output-lines INS_1 through INS_m for each machine instruction.
  - According to code in the IR, one of the output-lines INS_1 through INS_m is set to 1, and all other lines are set to 0.
- **Step-Decoder** provides a separate signal line for each step in the control sequence.
- **Encoder**
  - It gets the input from instruction decoder, step decoder, external inputs and condition codes.
  - It uses all these inputs to generate individual control-signals: Y_{in}, PC_{out}, Add, End and so on.
  - For example (Figure 7.12), $Z_{in}=T_1+T_6.ADD+T_4.BR$
    ;This signal is asserted during time-slot T_1 for all instructions.
    during T_6 for an Add instruction.
    during T_4 for unconditional branch instruction
- When **RUN**=1, counter is incremented by 1 at the end of every clock cycle.
  When RUN=0, counter stops counting.
- After execution of each instruction, **end** signal is generated. End signal resets step counter.
- Sequence of operations carried out by this machine is determined by wiring of logic circuits, hence the name "**hardwired**".
- **Advantage**: Can operate at high speed.
- **Disadvantages:**
  1) Since no. of instructions/control-lines is often in hundreds, the complexity of control unit is very high.
  2) It is costly and difficult to design.
  3) The control unit is inflexible because it is difficult to change the design.