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

Internal Assessment Test 1 – Sept. 2017

Sub:	Advanced Computer Architecture(ACA)					Sub Code:	10CS74	Branch:	CSE
Date:	21/9/2017	Duration:	90 min's	Max Marks:	50	Sem / Sec:	VII/A,B,C	OBE	

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1	Define the term CPI and derive the equation for finding the total processor clock cycles needed to execute a program. Consider the execution of an object code with 200,000 instructions on a 20 MHZ processor. The program consists of four major types of instruction. The instruction mix and number of cycles (CPI) needed for each instruction is given below. i) Find total number of cycles required to execute the program. ii) Calculate the average CPI when the program is executed on uniprocessor system with above trace results. iii) Calculate the MIPS rate based on CPI obtained in ii.	[10]	CO1	L3

No	Instruction type	CPI	Instruction mix
1	Arithmetic and logic	1	68%
2	Load/store with cache hit	2	8%
3	Branch	4	14%
4	Memory reference with cache hit	8	10%

2 (a)	Elaborate the different parameters that decide the cost of an IC. Give the equation of each parameter separately and explain them.	[06]	CO1	L2
(b)	Find the die yield for the dies that are 1.5cm on a side and 1.0 cm on a side, assuming a defect density of 0.4per cm^2 and α is 4.	[04]	CO1	L3
3	Define Computer Architecture and ISA. Explain seven dimensions of Instruction Set Architecture (ISA).	[10]	CO2	L1
4 (a)	Briefly explain Amdahl's law	[06]	CO1	L2
(b)	Suppose that we want to enhance the processor used for Web serving. The new processor is 10 times faster on computation in the Web serving application than the original processor. Assuming that the original processor is busy with computation 40% of the time and is waiting for I/O 60% of the time, what is the overall speedup gained by incorporating the enhancement?	[04]	CO1	L3
5 (a)	List and explain four important technologies which have change at a dramatic pace and are critical to modern implementation.	[05]	CO1	L1
(b)	Write short notes on benchmarks	[05]	CO1	L2
6	With a neat diagram explain the classic five stage pipeline for RISC processor	[10]	CO2	L2
7 (a)	List three hazards in Pipelining. Explain any one hazard briefly with neat diagram	[06]	CO2	L2
(b)	Consider the un-pipelined processor in the previous section. Assume that it has a 1 ns clock cycle and that it uses 4 cycles for ALU operations and branches and 5 cycles for memory operations. Assume that the relative frequencies of these operations are 40%, 20%, and 40%, respectively. Suppose that due to clock skew and setup, pipelining the processor adds 0.2 ns of overhead to the clock. Ignoring any latency impact, how much speedup in the instruction execution rate will we gain from a pipeline?	[04]	CO2	L3
8	Explain the basic MIPS pipeline with neat diagram	[10]	CO2	L2

Scheme and Solution : Internal Assessment Test 1 – Sept. 2017

Sub:	**Advanced Computer Architecture(ACA)**					Sub Code:	10CS74	Branch:	CSE
Date:	21/9/2017	Duration:	90 min's	Max Marks:	50	Sem / Sec:	VII/A,B,C		

Scheme

1. Definition of CPI (1 mark).
   Derivation of equation for total processor clock cycles (3 marks).
   Solving the problem (6 marks).
2. A) Impact of time, volume and commodification on cost of IC (2 marks).
   Explain the Formula for cost of IC, Cost of die, Die yield and Dies per wafer (4 marks).
2. B) Formula (1 mark).
   Solving the problem for die with 1.5cm side and 1 cm side (3 marks).
3. Define Computer Architechture (1 mark).
   Define ISA (2 marks).
   Explanation on Seven dimensions of ISA (7 marks).
4. A) Explanation and derivation of formula for speedup (6 marks).
4. B) Formula for speedup (1 mark).
   Calculation of speedup (3 marks).
5. A) List all 4 technologies (1 mark).
   Explain each technology (4 marks).
5. B) Define benchmark(1M)
   Explanation on Desktop, Server, SPEC and TP benchmarks (4 marks)
6. Diagram of RISC pipeline (3 marks).
   Explanation on 5-stage RISC pipeline (7 marks).
7. A) Listing the pipeline hazards (1 mark).
   Explanation of the hazard (3 marks).
   Diagram (2 marks).
7. B) Formula (1 mark)

Calculation of average execution time for pipelined and un-pipelined (2 marks).

Calculation of Speedup (1 mark).

8. Diagram of MIPS pipeline (3 marks).
   Explanation (7 marks).

Solution

1. In computer architecture, **cycles per instruction** (**clock cycles per instruction**, **clocks per instruction**, or CPI) is one aspect of a processor's performance: the average number of **clock cycles per instruction** for a program or program fragment. It is the multiplicative inverse of **instructions per cycle**.

   Essentially all computers are constructed using a clock running at a constant rate. These discrete time events are called *ticks, clock ticks, clock periods, clocks, cycles,* or *clock cycles.* Computer designers refer to the time of a clock period by its duration (e.g., 1 ns) or by its rate (e.g., 1 GHz). CPU time for a program can then be expressed two ways:

   $$\text{CPU time} = \text{CPU clock cycles for a program} * \text{Clock cycle time}$$

   $$\text{CPU time} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

   If we know the number of clock cycles and the instruction count, we can calculate the average number of *clock cycles per instruction* (CPI). Because it is easier to work with, and because we will deal with simple processors in this chapter, we use CPI. Designers sometimes also use *instructions per clock* (IPC), which is the inverse of CPI.

   CPI is computed as

   $$\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count(IC)}}$$

By transposing instruction count in the above formula, clock cycles can be defined as IC × CPI.

This allows us to use CPI in the execution time formula:

CPU time = Instruction count(IC) × Cycles per instruction × Clock cycle time

Hence the total processor clock cycles are given below

i) The formula for finding the total processor clock cycles is given below

$$\text{CPU clock cycles} = \sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i$$

CPU Clock cycles= [(68% * 200000) *1] + [(8% * 200000)*2] + [(14%*200000)*4]

+ [(10% * 200000)*8]

= 440000

ii) The formula for finding the total CPI is given below

$$\text{CPI} = \frac{\sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i}{\text{Instruction count}} = \sum_{i=1}^{n} \frac{\text{IC}_i}{\text{Instruction count}} \times \text{CPI}_i$$

$\text{CPI} = \sum \text{Frequency}_i * \text{CPI}_i$

$\text{Frequency}_i = \text{IC}_i$ / Instruction count

CPI = (0.68*1) + (0.08x2) + (0.14*4) + (0.1*8)

= 2.2

ii) The formula for finding the MIPS rate is given below

MIPS rate = clock rate / (CPI $*10^6$)

MIPS rate = ($20*10^6$) / ($2.2* 10^6$)

= 9.09

2. A) Typically IC is produced in large batches on a single wafer of EGS (Electronic Grade Silicon) or other semiconductor through process such as photolithography. The wafer is cut into many pieces each containing one copy of the circuit. Each of these pieces is called die. A die is small block of semi conducting material on which a given functional circuit is fabricated. Wafer provides the base for the IC. IC is mounted on wafer. Thus the cost of the integrated circuit is given by the formula

$$\text{Cost of integrated circuit} = \frac{\text{Cost of die} + \text{Cost of testing die} + \text{Cost of packaging and final test}}{\text{Final test yield}}$$

Thus the cost of IC is dependent on cost of die, cost of testing die and cost of final packaging. Final test yield is calculated as number of units which survived the testing procedure divided by the total number of units tested.

The cost of die is calculated by the formula as given below

$$\text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per wafer} \times \text{Die yield}}$$

The cost of die is dependent on the cost of wafer, number of dies which will fit on a wafer and also die yield. Die yield corresponds to percentage of good dies in the wafer. The number of good dies per wafer is approximately the area of the wafer divided by the area of the die. It is given as follows

$$\text{Dies per wafer} = \frac{\pi \times (\text{Wafer diameter}/2)^2}{\text{Die area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2 \times \text{Die area}}}$$

The number of dies per wafer is approximately the area of the wafer divided by the area of the die. The first term is the ratio of wafer area ($\pi r2$) to die area. The second term compensates for rectangular dies near the periphery of round wafers. Dividing the circumference (πd) by the diagonal of a square die is approximately the number of dies along the edge.

However, this only gives the maximum number of dies per wafer. The critical question is: What is the fraction of good dies on a wafer number, or the *die yield*? A simple model of integrated circuit yield, which assumes that defects are randomly distributed over the wafer and that yield is inversely proportional to the complexity of the fabrication process, leads to the following.

$$\text{Die yield} = \text{Wafer yield} \times \left(1 + \frac{\text{Defects per unit area} \times \text{Die area}}{\alpha}\right)^{-\alpha}$$

Wafer yield accounts for wafers that are completely bad and so need not be tested. For simplicity, we'll just assume the wafer yield is 100%.Defects per unit area is a measure of the random manufacturing defects that occur. Lastly, α is a parameter which is a measure of manufacturing complexity.

2. B) ***Answer*** The total die areas are 2.25 cm^2 and 1.00 cm^2. For the larger die, the yield is

$$\text{Die yield} = \left(1 + \frac{0.4 \times 2.25}{4.0}\right)^{-4} = 0.44$$

For the smaller die, it is $\text{Die yield} = \left(1 + \frac{0.4 \times 1.00}{4.0}\right)^{-4} = 0.68$

That is, less than half of all the large die are good but more than two-thirds of the small die are good.

3. **Definition of Computer Architecture:** In computer engineering, computer architecture is a set of rules and methods that describe the functionality, organization, and implementation of computer systems.

   Computer Architecture is the art of selecting and interconnecting hardware components to create the computer that meets functional, performance and cost goals and formal modeling of those systems.

   Definition of ISA: Instruction set Architecture is the interface between the software and hardware. ISA defines the instructions, data types, registers, addressing modes which are visible to the programmer.

   Seven dimensions of ISA

   1. Class of ISA
   2. Memory Addressing
   3. Addressing modes
   4. Type and size of operands
   5. Operations
   6. Control flow instruction
   7. Encoding an ISA

Class of ISA

1. There are two important classes of GPR based ISA.
2. Register-memory ISA such as 80*86: Here one input operand (i.e. data or value) is in register and other input operand in memory and after ALU operation result is stored in register.
3. Load-store ISA such as MIPS: Here one input operand is in register and other input operand is in register and result goes to register. To transfer the value to memory we need to use load store instructions.
4. The 80*86 has16 general purpose registers and 16 floating point registers.
5. MIPS has 32 general purpose registers and 32 floating-point register.

Memory Addressing

1. The 80*86 and MIPS use byte addressing for accessing memory operands.
2. Byte addressing refers to architectures where data can be accessed 8 bits (1 Byte) at a time. In some machines; accesses to objects larger than a byte must be aligned.
3. For MIPS the data is aligned and for 8086 data is misaligned.
4. An access to an object of size s bytes at byte address A is aligned if A mod s = 0. Misalignment causes hardware complications.

Addressing modes

1. Addressing modes specify the address of a memory object. An addressing mode specifies how to calculate the effective memory address of an operand by using information held in registers and/or constants contained within a machine instruction or elsewhere.
2. MIPS addressing modes are Register, Immediate (for constants), and Displacement, where a constant offset is added to a register to form the memory address. The 80x86 supports those three plus three variations of displacement: no register (absolute or direct addressing mode), two registers (based indexed with displacement), two registers where one register is multiplied by the size of the operand in bytes (based scaled index and displacement)

Type and Size of Operands

1. Like most of the ISAs the 8086 and MIPS support operand size of 8-bit (ASCII Character), 16-bit (Unicode character or half word), 32-bit (Integer or word), 64-bit

(Long or double word) and floating point in 32-bit (Single precision) and 64-bit (Double precision).

2. The 80x86 also supports 80-bit floating point (extended double precision).

Operations

1. The general categories of operations are data transfer, arithmetic, logical, control, and floating point operations.
2. The instructions for data transfer are LB,LW,SB,SW,SD etc.
3. The instructions for arithmetic operations are DADD,DADDI,DMUL,DDIV etc.
4. The instructions for control flow operations are BEQ,BNEZ etc.

Control Flow Instructions

1. Virtually all ISAs including 8086 and MIPS support conditional branches, unconditional jumps, procedure calls, and returns.
2. There are some small differences. MIPS conditional branches (BE,BNE, etc.) test the contents of registers, while the 80x86 branches (JE,JNE, etc.) test condition code bits set as side effects of arithmetic/logic operations.
3. MIPS procedure call (JAL) places the return address in a register, while the 80x86 call (CALLF) places the return address on a stack in memory.

Encoding an ISA

1. Encoding means converting the instruction into binary form. There are two types of encoding 1) fixed length encoding 2) variable length encoding.
2. The MIPS has fixed length encoding and all instructions are 32-bit long.
3. While 8086 has Variable length encoding and instructions are ranging from 1-18 bytes, so a program compiled for the 80x86 is usually smaller than the same program compiled for MIPS.

4. A) The performance gain that can be obtained by improving some portion of a computer can be calculated by Amdahl's Law. Amdahl's law defines the speedup that can be gained by using a particular feature. Speedup is the ratio

$$\text{Speedup} = \frac{\text{Performance for entire task using the enhancement}}{\text{Performance for entire task without using the enhancement}}$$

$$\text{Speedup} = \frac{\text{Execution time for entire task without using the enhancement}}{\text{Execution time for entire task using the enhancement}}$$

Speedup tells how much faster a task will run using the computer with the enhancement as opposed to the original computer.

Amdahl's Law gives us a quick way to find the speedup from some enhancement, which depends on two factors:

The fraction of the computation time in the original computer that can be converted to take advantage of the enhancement—For example, if 20 seconds of the execution time of a program that takes 60 seconds in total can use an enhancement, the fraction is 20/60. This value, which we will call $\text{Fraction}_{\text{enhanced}}$, is always less than or equal to 1.

The improvement gained by the enhanced execution mode; that is, how much faster the task would run if the enhanced mode were used for the entire program—This value is the time of the original mode over the time of the enhanced mode. If the enhanced mode takes, say, 2 seconds for a portion of the program, while it is 5 seconds in the original mode, the improvement is 5/2. We will call this value, which is always greater than 1, $\text{Speedup}_{\text{enhanced}}$.

The overall speedup is the ratio of the execution times:

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

4. B) *Answer* $\text{Fraction}_{\text{enhanced}} = 0.4, \text{Speedup}_{\text{enhanced}} = 10, \text{Speedup}_{\text{overall}} = \frac{1}{0.6 + \frac{0.4}{10}} = \frac{1}{0.64} \approx 1.56$

5. A) Four implementation technologies, which change at the dramatic pace, are as follows.
   - Integrated Circuit logic technology: An integrated circuit or monolithic integrated circuit is a set of electronic circuits on one small plate ("chip") of semiconductor material, normally silicon. Also thousands and millions of tiny registers, capacitors

and transistors are fabricated in IC. Transistor density increases by about 35% per year, quadrupling in somewhat over four years. Increases in die size are less predictable and slower, ranging from 10% to 20% per year.

- Semiconductor DRAM: It is used for main memory and is very cheap. It contains many memory cells and each contains one transistor and capacitor to store 1 bit of data. Capacity increases by about 20 to 40% per year, doubling roughly every two years.
- Magnetic Disk technology: Examples of magnetic storage media include floppy disks, hard disks, hard drives, magnetic recording tape, and magnetic stripes on credit cards. Prior to 1990, density increased by about 30% per year, doubling in three years. It rose to 60% per year thereafter, and increased to 100% per year in 1996. Since 2004, it has dropped back to 30% per year. Disks are still 50–100 times cheaper per bit than DRAM.
- Network technology: Network performance depends on the performance of switches and on performance of transmission system. A network switch connects devices together on a computer network and uses packet switching to receive, process and forward data to the destination device. The most useful LAN technology is Ethernet. Table below shows different Ethernet standards.

Local area network	Ethernet	Fast Ethernet	Gigabit Ethernet	10 Gigabit Ethernet
IEEE standard	802.3	803.3u	802.3ab	802.3ac
Year	1978	1995	1999	2003
Bandwidth (MBit/sec)	10	100	1000	10000
Latency (μsec)	3000	500	340	190

For all the technologies the cost decreases at about the rate at which density increases.

5. B) Benchmark is a standard which forms the basis of measurement and through which the performance of others could be judged. The best choice of benchmark to measure the performance is the compiler itself. If we consider smaller programs as benchmarks but it will have many pitfalls. Examples include
   - Kernels which are small, key pieces of real applications. Example: Livermore loops, LINPACK.

- Toy programs, which are 100 line programs for example: quick sort
- Synthetic benchmarks, which are fake programs invented to try to match the profile and behavior of real applications, such as Dhrystone.

Thus nowadays various benchmark suites are available to measure the performance for variety of systems. Nonetheless, a key advantage of such suites is that the weakness of any one benchmark is lessened by the presence of the other benchmarks.

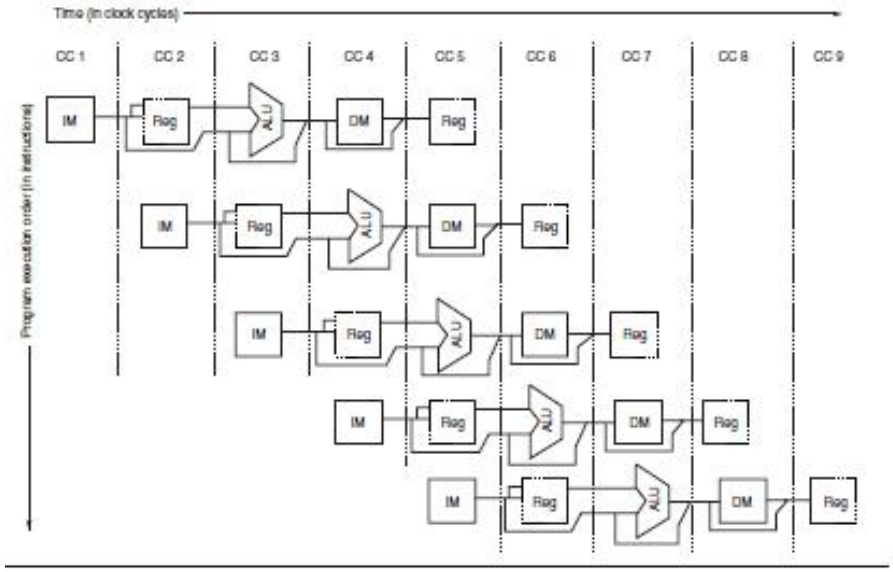
- Desktop benchmarks divide into two broad classes: processor-intensive benchmarks and graphics-intensive benchmarks.
- SPEC originally created a benchmark set focusing on processor performance (initially called SPEC89), which has evolved into its fifth generation: SPEC CPU2006, which follows SPEC2000, SPEC95, SPEC92, and SPEC89.
- SPEC benchmarks are real programs modified to be portable and to minimize the effect of I/O on performance.
- The simplest benchmark used here is processor throughput benchmark. For example : SPECCPU2000 is a processor throughput benchmark which is used to measure the processing rate by running the multiple copies of each SPEC CPU benchmark on multiple processors.
- Transaction-processing (TP) benchmarks measure the ability of a system to handle transactions, which consist of database accesses and updates. Airline reservation systems and bank ATM systems are typical simple examples of TP systems in which this benchmark is needed.

6. **Classic five stage pipeline for RISC processor**
    - Every instruction in the RISC can be implemented in 5 clock cycles. The five clock cycles are IF, ID, EX, MEM and WB.
    - Although each instruction will take 5 clock cycles for its execution the hardware will initiate a new instruction in every clock cycle and will be executing some part of different instructions.
    - The basic RISC pipeline is shown below. On each clock cycle another instruction is fetched and begins its 5-cycle execution.

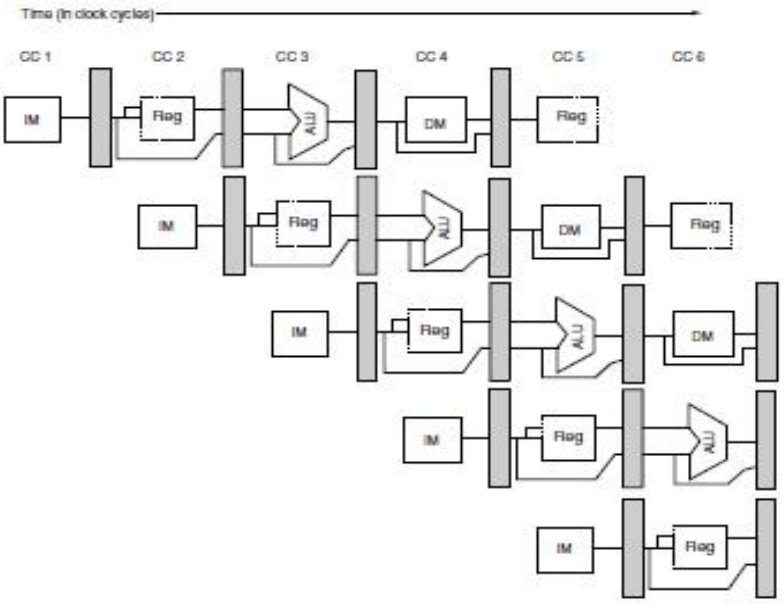
	Clock number								
Instruction number	1	2	3	4	5	6	7	8	9
Instruction i	IF	ID	EX	MEM	WB				
Instruction $i+1$		IF	ID	EX	MEM	WB			
Instruction $i+2$			IF	ID	EX	MEM	WB		
Instruction $i+3$				IF	ID	EX	MEM	WB	
Instruction $i+4$					IF	ID	EX	MEM	WB

Simple RISC pipeline. On each clock cycle, another instruction is fetched and begins its 5-cycle execution. If an instruction is started every clock cycle, the performance will be up to five times that of a processor that is not pipelined. The names for the stages in the pipeline are the same as those used for the cycles in the unpipelined implementation: IF = instruction fetch, ID = instruction decode, EX = execution, MEM = memory access, and WB = write back.

- But the overlapped execution of multiple instructions inside the pipeline should not introduce new type of conflicts.
- Hence to avoid conflicts three methods are used in a pipeline.
- First we separate the instruction and data memory which we would typically implement with separate instruction and data caches. The use of separate caches eliminates a conflict for a single memory that would arise between IF(Instruction Fetch) stage and memory access(MEM) stage.
- Second the register file is used in two stages one for reading in ID and one for writing in WB. Thus to handle reading and writing to same register perform the register write in first half of the clock cycle and register read in the second half of the clock cycle. It is indicated by solid and dashed lines in the figure given below.
- Figure shows the pipeline and abbreviation IM is used for instruction memory and DM for data memory and CC for clock cycle.



- The PC should be updated to start the execution of the new instruction in every clock cycle. PC points to the address of the next instruction to be fetched for execution. But for branches the PC is not changed until ID stage and it causes problem.
- Also pipeline registers are present between the successive stages of the pipeline so that all the results from a given stage are stored into register that is used as the input to the next stage on the next clock cycle.
- For example: The result of the ALU operation is computed during the EX stage, but not actually stored until WB, it arrives there by passing through two pipeline registers. The pipeline registers are named as IF/ID, ID/EX, EX/MEM and MEM/WB.



A pipeline showing the pipeline registers between successive pipeline stages. Notice that the registers prevent interference between two different instructions in adjacent stages in the pipeline. The registers also play the critical role of carrying data for a given instruction from one stage to the other. The edge-triggered property of registers—that is, that the values change instantaneously on a clock edge—is critical. Otherwise, the data from one instruction could interfere with the execution of another!

7. A) There are three type of Hazards

Structural Hazard

Data Hazard

Branch Hazard

Data Hazard

Data hazard occurs when one instruction is dependent on the result produced by the other instruction.

Data hazards occur when the pipeline changes the order of read/write accesses to operands so that the order differs from the order seen by sequentially executing instructions on an unpipelined processor. Consider the pipelined execution of these instructions:

DADD R1,R2,R3

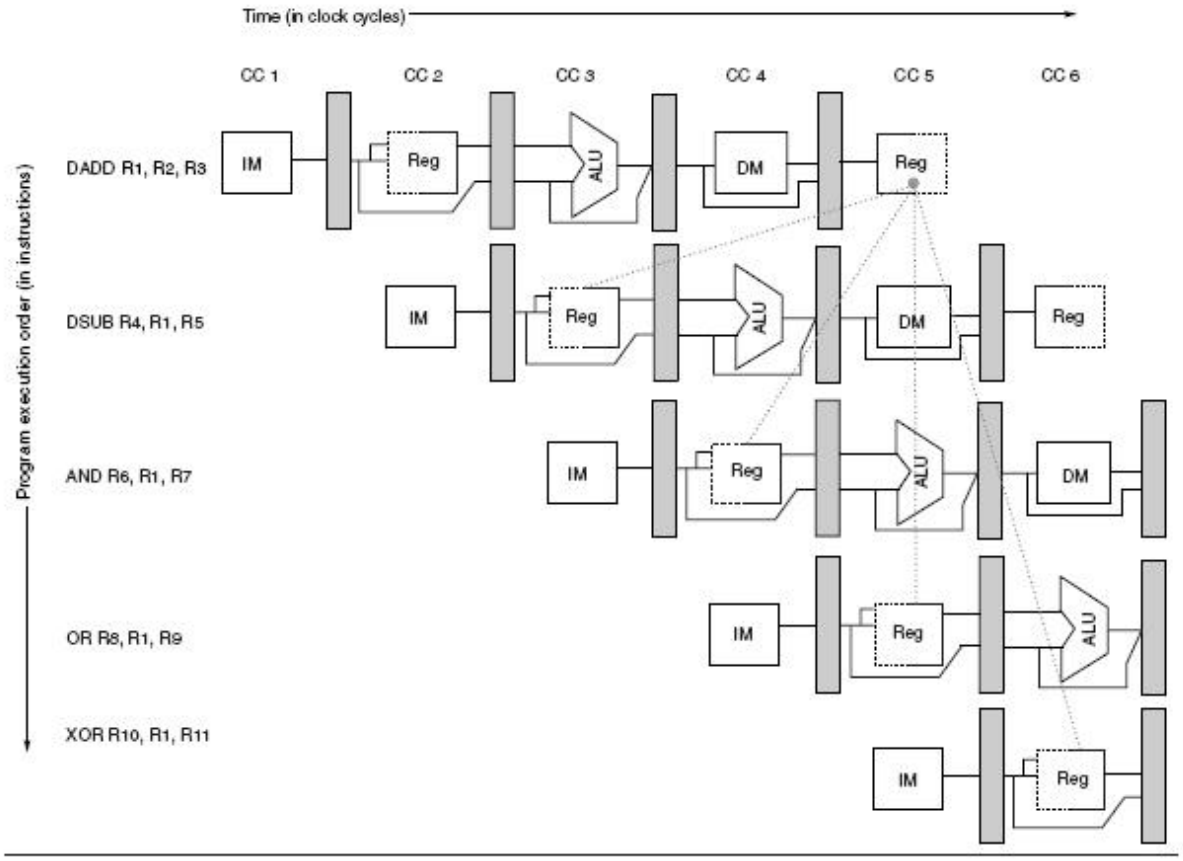
DSUB R4,R1,R5

AND R6,R1,R7

OR R8,R1,R9

XOR R10,R1,R11

All the instructions after the DADD use the result of the DADD instruction i.e. R1. As shown in Figure the DADD instruction writes the value of R1 in the WB pipe stage, but the DSUB instruction reads the value during its ID stage. This problem is called a *data hazard*.



The use of the result of the DADD instruction in the next three instructions causes a hazard, since the register is not written until after those instructions read it.

The AND instruction is also affected by this hazard. As we can see from Figure, the write of R1 does not complete until the end of clock cycle 5. Thus, the AND instruction that reads the registers during clock cycle 4 will receive the wrong results.

The XOR instruction operates properly because its register read occurs in clock cycle 6, after the register write. The OR instruction also operates without incurring a hazard because we perform the register file reads in the second half of the cycle and the writes in the first half.

7. B) The average instruction execution time on the unpipelined processor is

Average instruction execution time = Clock cycle x Average CPI

= 1 ns x ((40% + 20%) x 4 + 40% x 5)

= 1 ns x 4.4

= 4.4 ns

The average instruction execution time on the pipelined processor is

Average instruction execution time = Clock cycle x Average CPI

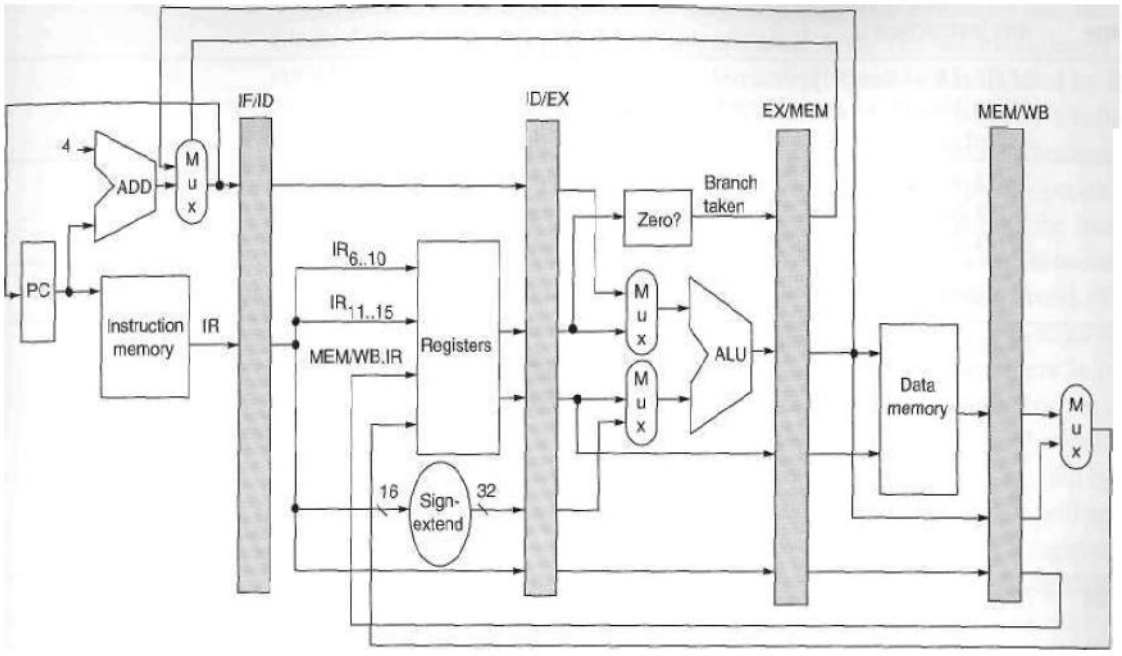
=(1 + 0.2) * 1

=1.2ns

$$\text{Speedup from pipelining} = \frac{\text{Average instruction time unpipelined}}{\text{Average instruction time pipelined}}$$

$$= \frac{4.4 \text{ ns}}{1.2 \text{ ns}} = 3.7 \text{ times}$$

8. A Basic Pipeline for MIPS



- In this implementation there are four pipeline registers namely IF/ID, ID/EX, EX/MEM and MEM/WB between different pipe stages. Instruction Memory contains the set of instructions which should be fetched for execution. After fetching the instruction is copied in IR. The registers are labeled with the names of the stages they connect. All of the registers needed to hold values temporarily between clock cycles within one instruction are subsumed into these pipeline registers. The pipeline registers carry both data and control from one pipeline stage to the next.
- The data is copied from one pipeline register to another until it is no longer needed. The value of PC is incremented by 4 in IF stage. The selection multiplexer in the IF stage selects the value of PC. Thus if the instruction is branch instruction it selects the PC of branch target address else it selects PC + 4.
- The zero test in EX stage is testing if the value of register is equal to zero.
- If the branch is taken branch then the PC is updated in EX stage and written in MEM cycle.
- Thus the branch instruction has penalty of 3 clock cycles which can be reduced by computing the PC earlier in the pipeline
- The two multiplexers in the EX stage are set depending on the instruction type.
- The top ALU input multiplexer is set by whether the instruction is a branch or not, and the bottom multiplexer is set by whether the instruction is a register-register ALU operation or any other type of operation
- The multiplexer in the IF stage chooses whether to use the value of the incremented PC or the value of the EX/MEM.ALUOutput (the branch target) to write into the PC. This multiplexer is controlled by the field EX/MEM.cond. Also EX/MEM.cond is set to 1 if the branch is taken branch.
- The fourth multiplexer is controlled by whether the instruction in the WB stage is a load or an ALU operation.
- Notice in figure below that we have made two important changes, each of which removes 1 cycle from the 3-cycle stall for branches.
- The first change is to move both the branch-target address calculation and the branch condition decision to the ID cycle.

- The second change is to write the PC of the instruction in the IF phase, using either the branch-target address computed during ID or the incremented PC computed during IF.
- Computing the branch-target address during ID requires an additional adder. With the separate adder and a branch decision made during ID, there is only a 1- clock-cycle stall on branches

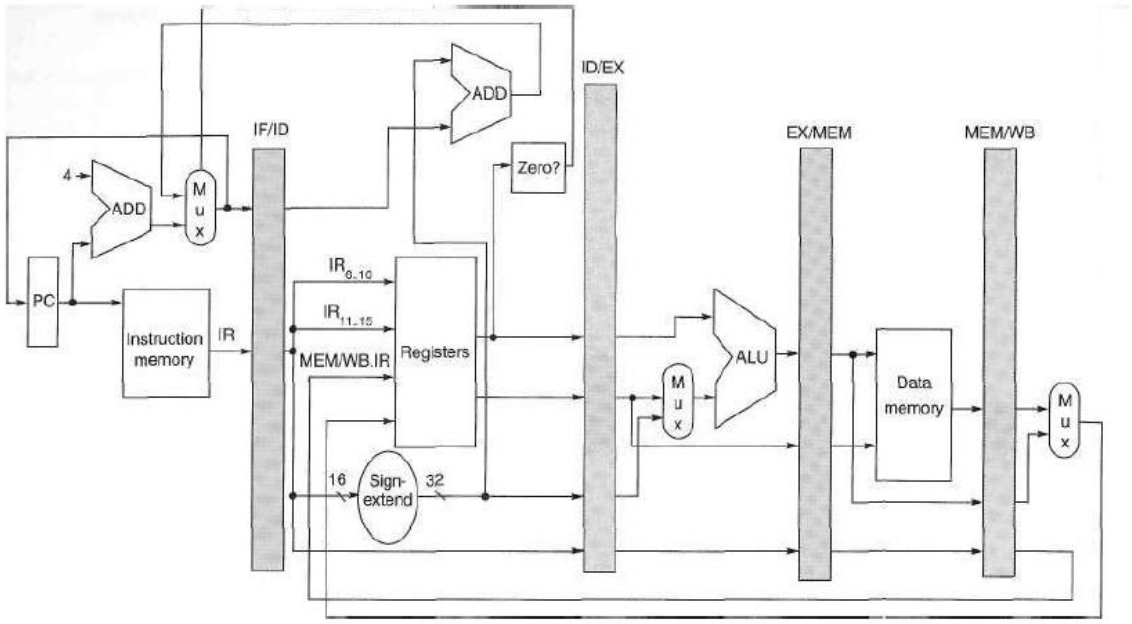


Figure A.24 The stall from branch hazards can be reduced by moving the zero test and branch-target calculation into the ID phase of the pipeline.