SNS COLLEGE OF TECHNOLOGY

(An Autonomous Institution)
Coimbatore – 641 035.

B.E / B.Tech – Internal Assessment Exam- I
Academic Year 2023-2024 (ODD)

Third Semester (Regulation R2019)

19ITT202 – Computer Organization and Architecture – Answer Key
(Common to CSE and IT)

TIME: 1.5 HOURS **MAXIMUM MARKS: 50**

ANSWER ALL QUESTIONS
PART A

1. Computer A has a clock cycle time of 250 ps and a CPI of 2.0 for some program, and computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program. Which computer is faster for this program and by how much?

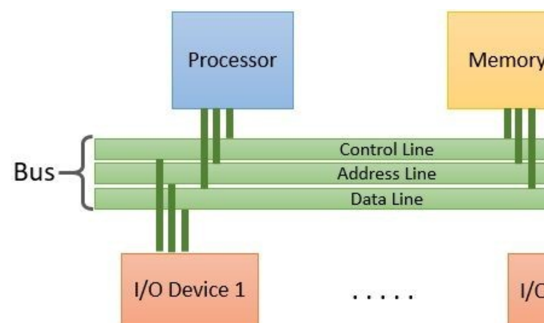
Assume each computer executes I instructions,
so
CPU timeA = I × 2.0 × 250 = 500 × I ps
CPU timeB = I × 1.2 × 500 = 600 × I ps
A is faster by the ratio of execution times:
performanceA / performanceB = execution timeB / execution timeA
= 600 × I / 500 × I
= 1.2
computer A is 1.2 times as fast as computer B for this program.

2. What is meant by straight line sequencing?

- Straight line sequencing means the instruction of a program is executed in a sequential manner(i.e. every time PC is incremented by a fixed offset).
- And no branch address is loaded on the PC.

3. What is meant by Bus Structure in Computer Architecture?

- Bus structures in computer plays important role in connecting the internal components of the computer.
- The bus in the computer is the shared transmission medium. This means multiple components or devices use the same bus structure to transmit the information signals to each other.



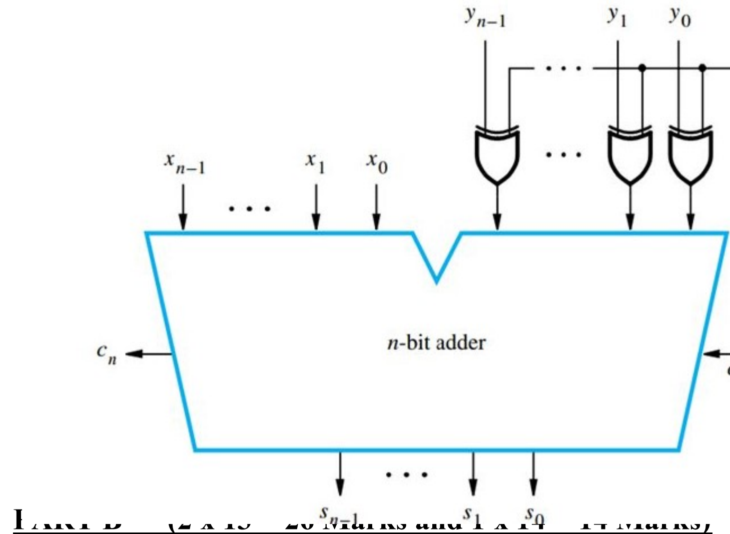
4. Consider two 8 bit positive number +98 and +87 and perform binary addition

01001 1000

01000 0111

10001 1111

5.Sketch the binary addition and subtraction logic Network



6. (a) (i) Formulate the CPU Performance equation and compose the various factors that affect performance

(ii) Formulate the CPU Performance equation and compose the various factors that affect performance

Response time is the time from start to completion of a task. This also includes:

- Operating system overhead.
- Waiting for I/O and other processes
- Accessing disk and memory
- Time spent executing on the CPU or execution time.

Throughput is the total amount of work done in a given time.
CPU execution time is the total time a CPU spends computing on a given task. It also excludes time for I/O or running other programs. This is also referred to as simply CPU time.
Performance is determined by execution time as performance is inversely proportional to execution time.
Performance = (1 / Execution time)
And,
(Performance of A / Performance of B) = (Execution Time of B / Execution Time of A)
The time to execute a given program can be computed as:
Execution time = CPU clock cycles x clock cycle time
Since clock cycle time and clock rate are reciprocals, so,
Execution time = CPU clock cycles / clock rate
The number of CPU clock cycles can be determined by,
CPU clock cycles = (No. of instructions / Program) x (Clock cycles / Instruction)

= Instruction Count x CPI

Which gives,

Execution time = Instruction Count x CPI x clock cycle time

= Instruction Count x CPI / clock rate

The units for CPU Execution time are

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{programs}} * \frac{\text{Clock Cycles}}{\text{Instructions}} * \frac{\text{Seconds}}{\text{Clock Cycles}}$$

Performance depends on

- Algorithm: affects IC, possibly CPI
- Programming language: affects IC, CPI
- Compiler: affects IC, CPI
- Instruction set architecture: affects IC, CPI, Tc

(ii) Explain in detail about different instruction types and instruction sequencing with your own example.

Four types of operations

1. Data transfer between memory and processor registers.
2. Arithmetic & logic operations on data
3. Program sequencing & control
4. I/O transfers.

1) Register transfer notations(RTN)

R3<–[R1]+[R2]

- Right hand side of RTN-denotes a value.
- Left hand side of RTN-name of a location.

2) Assembly language notations(ALN)

Add R1, R2, R3

- Adding contents of R1, R2 & place sum in R3.

3) Basic instruction types-4 types

- **Three address instructions**– Add A,B,C

A, B-source operands

C-destination operands

- **Two address instructions**-Add A,B

B <–[A] + [B]

- **One address instructions** –Add A

Add contents of A to accumulator & store sum back to accumulator.

- **Zero address instructions**

Instruction store operands in a structure called push down stack.

4) Instruction execution & straight line sequencing

- The processor control circuits use information in PC to fetch & execute instructions one at a time in order of increasing address.
- This is called straight line sequencing.
- Executing an instruction-2 phase procedures.
- 1st phase–**"instruction fetch"-**instruction is fetched from memory location whose address is in PC.
- This instruction is placed in instruction register in processor
- 2nd phase-**"instruction execute"-**instruction in IR is examined to determine which operation to be performed.

5) Branching

- Branch-type of instruction loads a new value into program counter.
- So processor fetches & executes instruction at this new address called "branch target"
- Conditional branch-causes a branch if a specified condition is satisfied.
- E.g. Branch>0 LOOP –conditional branch instruction .it executes only if it satisfies condition.

6) Condition codes

- Recording required information in individual bits called "condition code flags".
- These flags are grouped together in a special processor register called "condition code register" or "status register"
- Individual condition code flags-1 or 0.
- 4 commonly used flags.

1) N (negative)-set to 1 if result is –ve or else 0.
2) Z (zero)-set to 1 if result is 0, or else 0 .
3) V (overflow)-set to 1if arithmetic overflow occurs or else 0.
4) C(carry)-set to 1 if carry out results from operation or else 0

(OR)

(b) Define Addressing mode and explain the basic addressing modes with an example for each.

The addressing mode is the method to specify the operand of an instruction. The job of a microprocessor is to execute a set of instructions stored in memory to perform a specific task. Operations require the following:

- The operator or opcode which determines what will be done
- The operands which define the data to be used in the operation

The various types of Addressing modes are

i. Immediate mode
ii. Register mode
iii. Absolute mode
iv. Indirect mode
v. Index mode
vi. Base with index
vii. Base with index and offset
viii. Relative
ix. Auto increment
x. Auto decrement

Immediate mode: In this mode, the operand is specified in the instruction itself.

E.g. Move #200,R0

The above instruction places the value 200 in the register R0.Clearly Immediate mode can be used only to specify the source operand.

Figure 1 shows the above concept i.e. the operand is a part of the instruction

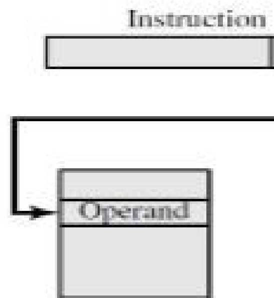
Register mode: The operand is the contents of a register. We specify the operand in this case by specifying the name of the register in the instruction. Processor registers are often used for

intermediate storage during arithmetic operations. This addressing mode is used at that time to access the registers.

E.g. Move R0, R1

Contents of the R0 register are moved to R1 register.

As shown in Figure 2, the instruction names the register that holds the operand.



Absolute mode: The operand is in a memory location; the address of the operand is passed explicitly in the instruction. Global variables are represented using this addressing mode.

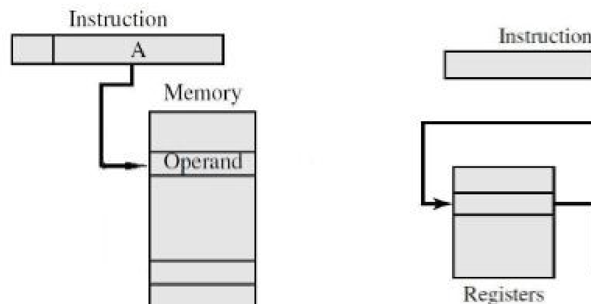
E.g. Move LOC, R0

Here LOC corresponds to the address from where the contents will be accessed by the processor and placed in R0.

Indirect mode: The effective address (E.A.) of the operands is the contents of a register (see Figure 3(b)) or the memory location whose address appears in the instruction (see Figure 3(a)). The name of the register or the memory address is placed in parentheses to denote indirection or in other words that the contents are addresses of the operands.

E.g. Add (R1), R0 (this mode is often called as register indirect mode) Add (B), R0

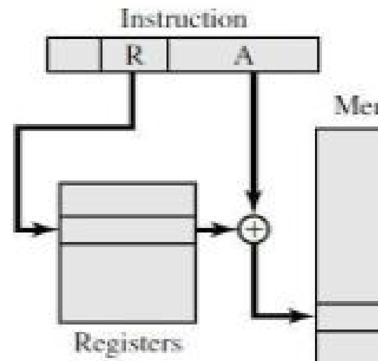
This instruction fetches the operand from the address, pointed by the contents of the register R1 or of the memory location ‘B’ and adds them to R0.



Index mode: The effective address of the operand is calculated by adding a constant value to the contents of a register, which is clearly shown in Figure 4. The address can be in a register used specially for this purpose or any of the general purpose registers. In either case it is called as an index register.

E.g. Move X (R0), R1

Here, Contents at address X+R0 are moved to R1 .X contains a constant value.



Base with index mode: The effective address is the sum of contents of two registers. The first register as before is called the index and the second register is called the base register. This mode provides more flexibility since both the components are registers and can thus be changed.

E.g. Move (R0, R1), R2

Here, Contents at address R0+R1 are moved to R2.

Base with index and offset mode: The effective address is the sum of contents of two registers and a constant. The constant value in this case is often called the offset or the displacement.

E.g. Move X (R0, R1), R2

Contents at address X+R0+R1 are moved to R2.

Relative mode: For relative addressing, also called PC-relative addressing, the implicitly referenced register is the program counter (PC). That is, the next instruction address is added to the address field to produce the EA. typically, the address field is treated as a twos complement number for this operation. Thus, the effective address is a displacement relative to the address of the instruction as shown in the example below.

Relative addressing exploits the concept of locality.

E.g. Move X (PC), R1

Here, Contents at address X+PC are moved to R1 .X contains a constant value.

Auto increment mode: The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to the next value. This increment is 1 for byte sized operands, 2 for 16 bit operands and so on.

E.g. Add (R2) +, R0

Here are the contents of R2 are first used as an E.A. then they are incremented.

Auto decrement mode: The effective address of the operand is the contents of a register specified in the instruction. Before accessing the operand, the contents of this register are automatically decremented and then the value is accessed.

E.g. Add - (R2), R0

Here are the contents of R2 are first decremented and then used as an E.A. for the operand which is added to the contents of R0. The auto increment addressing mode and the auto decrement addressing mode are widely used for the implementation of data structures like Stack. There may be other addressing modes that are unique to some processors. However,

the addressing modes mentioned above are common to many of the popular processors out there.

7. (a) Explain in detail about addition and subtraction of signed number with diagram and example.

signed-magnitude method is used by computers to implement floating-point operations. Signed-2's complement method is used by most computers for arithmetic operations executed on integers. In this approach, the leftmost bit in the number is used for signifying the sign; 0 indicates a positive integer, and 1 indicates a negative integer. The remaining bits in the number supported the magnitude of the number.

Example: -2410 is defined as −10011000

In this example, the leftmost bit 1 defines negative, and the magnitude is 24.

The magnitude for both positive and negative values is the same, but they change only with their signs. The range of values for the sign and magnitude representation is from -127 to 127.

There are eight conditions to consider while adding or subtracting signed numbers. These conditions are based on the operations implemented and the sign of the numbers. The table displays the algorithm for addition and subtraction. The first column in the table displays these conditions. The other columns of the table define the actual operations to be implemented with the magnitude of numbers. The last column of the table is needed to avoid a negative zero. This defines that when two same numbers are subtracted, the output must not be - 0. It should consistently be +0. In the table, the magnitude of the two numbers is defined by P and Q.

Addition and Subtraction of Signed Magnitude Numbers

Operations	Addition of Magnitudes	Subtraction of Magnitudes		
		P>Q	P<Q	P=Q
(+P) + (+Q)	+(P+Q)			
(+P) + (-Q)		+(P-Q)	-(Q-P)	+(P-Q)
(-P) + (+Q)		-(P-Q)	+(Q-P)	+(P-Q)
(-P) + (-Q)	-(P+Q)			
(+P) - (+Q)		+(P-Q)	-(Q-P)	+(P-Q)
(+P) - (-Q)	+(P+Q)			
(-P) - (+Q)	-(P+Q)			
(-P) - (-Q)		-(P-Q)	+(Q-P)	+(P-Q)

As display in the table, the addition algorithm states that −

- When the signs of P and Q are equal, add the two magnitudes and connect the sign of P to the output.
- When the signs of P and Q are different, compare the magnitudes and subtract the smaller number from the greater number.
- The signs of the output have to be equal as P in case P > Q or the complement of the sign of P in case P < Q.
- When the two magnitudes are equal, subtract Q from P and modify the sign of the output to positive.

The subtraction algorithm states that −

- When the signs of P and Q are different, add the two magnitudes and connect the signs of P to the output.
- When the signs of P and Q are the same, compare the magnitudes and subtract the smaller number from the greater number.

- The signs of the output have to be equal as P in case P > Q or the complement of the sign of P in case P < Q.
- When the two magnitudes are equal, subtract Q from P and modify the sign of the output to positive.

(OR)

(b) Illustrate the concept of Carry Look ahead Adder with diagram.

A carry-look ahead adder (CLA) is a type of adder used in digital logic. A carry-look ahead adder improves speed by reducing the amount of time required to determine carry bits. The carry-look ahead adder calculates one or more carry bits before the sum, which reduces the wait time to calculate the result of the larger value bits.

Operation Mechanism Carry look ahead depends on two things:

- Calculating, for each digit position, whether that position is going to propagate a carry if one comes in from the right.
- Combining these calculated values to be able to deduce quickly whether, for each group of digits, that group is going to propagate a carry that comes in from the right.

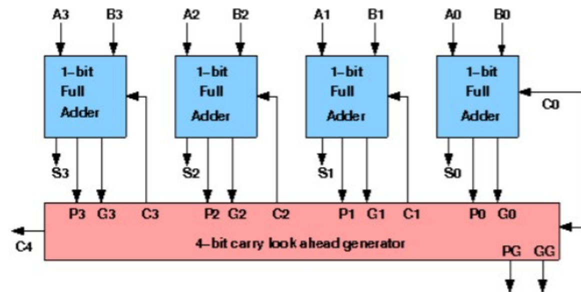
CLA – Concept

To reduce the computation time, there are faster ways to add two binary numbers by using carry lookahead adders.

They work by creating two signals P and G known to be Carry Propagator and Carry Generator.

The carry propagator is propagated to the next level whereas the carry generator is used to generate the output carry regardless of input carry.

The block diagram of a 4-bit Carry Lookahead Adder is shown here below



The number of gate levels for the carry propagation can be found from the circuit of full adder. The signal from input carry C_{in} to output carry C_{out} requires an AND gate and an OR gate, which constitutes two gate levels. So if there are four full adders in the parallel adder, the output carry C_5would have 2 X 4 = 8 gate levels from C_1 to C_5. For an n-bit parallel adder, there are 2n gate levels to propagate through.

Design Issues

The corresponding Boolean expressions are given here to construct a carry lookahead adder. In the carry-lookahead circuit we need to generate the two signals carry propagator(P) and carry generator(G),

- $Pi = Ai \oplus Bi$
- $Gi = Ai \cdot Bi$

The output sum and carry can be expressed as

$Sum_i = P_i \oplus C_i$

$C_{i+1} = G_i + (P_i \cdot C_i)$

Having these we could design the circuit. We can now write the Boolean function for the carry output of each stage and substitute for each C_i its value from the previous equations:

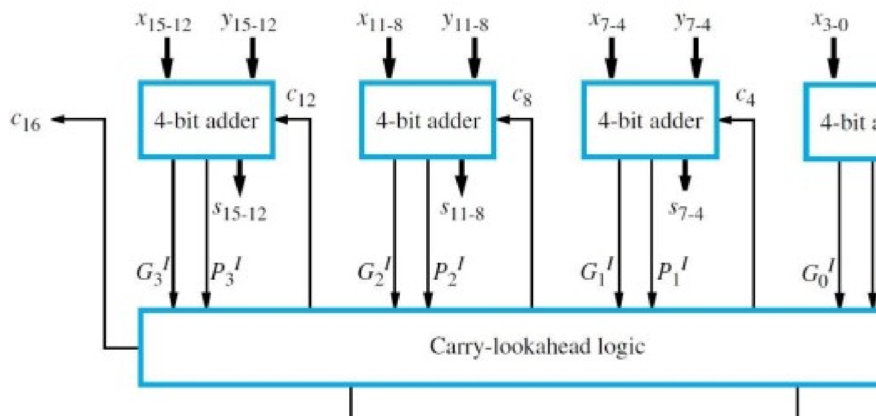
$$C_1 = G_0 + P_0 \cdot C_0$$

$$C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$$

$$C_3 = G_2 + P_2 \cdot C_2 = G_2\ P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

$$C_4 = G_3 + P_3 \cdot C_3 = G_3\ P_3 \cdot G_2\ P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

16 Bit – Carry lookahead adder



Advantages

- The propagation delay is reduced.
- It provides the fastest addition logic.

Disadvantages

- The Carry Look-ahead adder circuit gets complicated as the number of variables increase.
- The circuit is costlier as it involves more number of hardware.

8.(a)Assume that the variables f, g, h, i, and j are assigned to registers $s0, $s1, $s2, $s3, and $s4, respectively. Assume that the base address of the arrays A and B are in registers $s6 and $s7, respectively.

C Code: f = g + A[B[4]- B[3]];

For the C statement above, what is the corresponding MIPS assembly code?

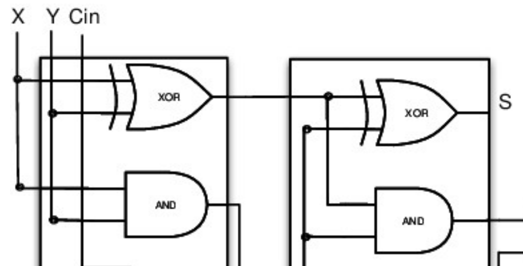
```
lw $t0, 16($s7) // $t0 = B[4]
lw $t1, 12($s7) // $t1 = B[3]
sub $t0, $t0, $t1 // $t0 = B[4] – B[3]
sll $t0, $t0, 2 // $t0 = $t0 * 4
add $t0, $t0, $s6 // $t0 = &A[B[4] – B[3]]
lw $t1, 0($t0) // $t1 = A[B[4] – B[3]]
```

add $s0, $s1, $t1 // f = g + A[B[4] – B[3]

OR

8. b Show how to implement full adder by using two half adders and external logic gates.

2 Half Adders and a OR gate is required to implement a Full Adder.



Inputs			Outp	
A	B	C_{in}	Sum	
0	0	0	0	
0	0	1	1	
0	1	0	1	
0	1	1	0	
1	0	0	1	
1	0	1	0	
[illegible]	[illegible]	[illegible]	[illegible]	