Computer Architecture I Final Exam

Chinese Name: _____

Pinyin Name: _____

Student ID: _____

E-Mail ... @shanghaitech.edu.cn: _____

Question:	1	2	3	4	5	6	7	8	9	10	11	Total
Points:	1	12	10	2	6	11	4	16	17	8	13	100
Score:												

- This test contains 21 numbered pages, including the cover page, printed on both sides of the sheet.
- Unless told otherwise always assume a 32bit machine.
- The total estimated time is 90 minutes.
- You have 90 minutes to complete this exam, 10 min for submission. The exam is open book.
- Under no circumstances are you allowed to communicate to anybody except the CS110 professors, inviligators or TAs.
- There may be partial credit for incomplete answers; write as much of the solution as you can. We will deduct points if your solution is far more complicated than necessary.
- Write your answers on blank A4 pages. Include the correct header on every page (see Question 1 below). At the end scan/ photo all pages into a pdf (there are plenty of apps for that). Submit in time - we will NOT accept late submissions!
- Each question from Q2 to Q11 will need to be started on a new A4 sheet - so you'll need at least 10 A4 sheets!

[1] 1. **Submitted pages format**

Correctly write the header for ALL the pages that you submit. That includes: Your name in Chinese, your ShanghaiTech Email address (without '@shanghaitech.edu.cn'), the string "CS110 Final", this Exam ID (see on the top left of the following pages), the page number (your own numbering). Also follow the rule that every new major question (e.g. Question 2, Question 3, etc.) is written on a new A4 paper. Submit correctly on gradescope or (only if gradescope fails) via email to soerensch@shanghaitech.edu.cn AND wangchd@shanghaitech.edu.cn .

2. **RISC-V**

Look at the following C program, you want to compile it to RISC-V instructions, but your 32-bit CPU is broken, it cannot execute sub and mul. So you decide to convert it by hand.

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int* i = malloc(sizeof(int));
    scanf("%d",i);
    printf("%d",abs(*i));
    free(i);
    return 0;
}

int abs(int a) {
    if(a < 0) a = -a;
    return a;
}
```

The following RISC-V code is the output.

```
.data
    format:  .asciiz "%d"
.text
.global main
main:
    # get memory for input and scan it
    0x00400513          # Translate this instruction into assembly
    jal malloc          # calling malloc(sizeof(int))
    mv s0, a0           # save returned address to s0
    _____________
    mv a1, s0           # set argument2, a1 == i
    jal scanf           # call scanf("%d",i), in stdio.s
    ____________        # get i and calculate the absolute value
    jal abs
    # print it
    mv s1, a0           # save returned abs value to s1
    _____________       # same as line 10
    mv a1, s1           # set argument2, a1 == returned abs value
    jal printf          # call printf("%d",abs(*i)), in stdio.s
    mv a0, s0
    jal free
    li a0, 10
    ecall               # return 0
abs:
    0x00055663          # Translate this instruction into assembly
    _____________
    _____________
done:
    jr ra
```

4 (a) Translate instructions to machine code written in hexadecimal, and translate machine code to RISC-V instructions!

line 9: `mv s0, a0` _____

line 29: `jr ra` _____

line 7: `0x00400513` _____

line 25: `0x00055663` _____

Solution:

```
1 0x00050413          #
2 0x0000806           #
3 li a0, 4            # (or addi x10, x0, 4)
4 bge a0, zero, done  # (or bge x10, x0, 12)
```

8 (b) Finish the code with blank lines, remember that your CPU cannot execute sub and mul.

line 10/17: _____

line 13: _____

line 26: _____

line 27: _____

Solution:

```
1 line 10/17: la a0, format
2 line 13:    lw a0, 0(s0)
3 line 26:    xori a0, a0, -1
4 line 27:    addi a0, a0, 1
```

3. **CALL**

8 (a) **(True or False)** Please fill your answer (T or F) in the table below.

1. A two-pass assembler builds the symbol table for all labels during the second pass.
2. When you compile your program, the compiler identifies the logic errors and suggests how to correct them.
3. Assembler is not allowed to have input containing pseudo instructions.

4. In general, an interpreter is closer to high-level at the expense of lower performance and larger code size compared to a translator.
5. A function of the linker is to replace absolute references in an object file by symbolic references to locations in other object files.
6. A static linker adds object files and libraries directly into the executable file. Any updates to object files and libraries externally will not affect the file.
7. Before branching to a start-up routine, the loader will initialize the processor registers and set the stack pointer to the first free stack location.
8. A function of the linker is to combine several object files into a single executable file.

1	2	3	4	5	6	7	8

Solution: F; F; F; F; F; T; T; T.

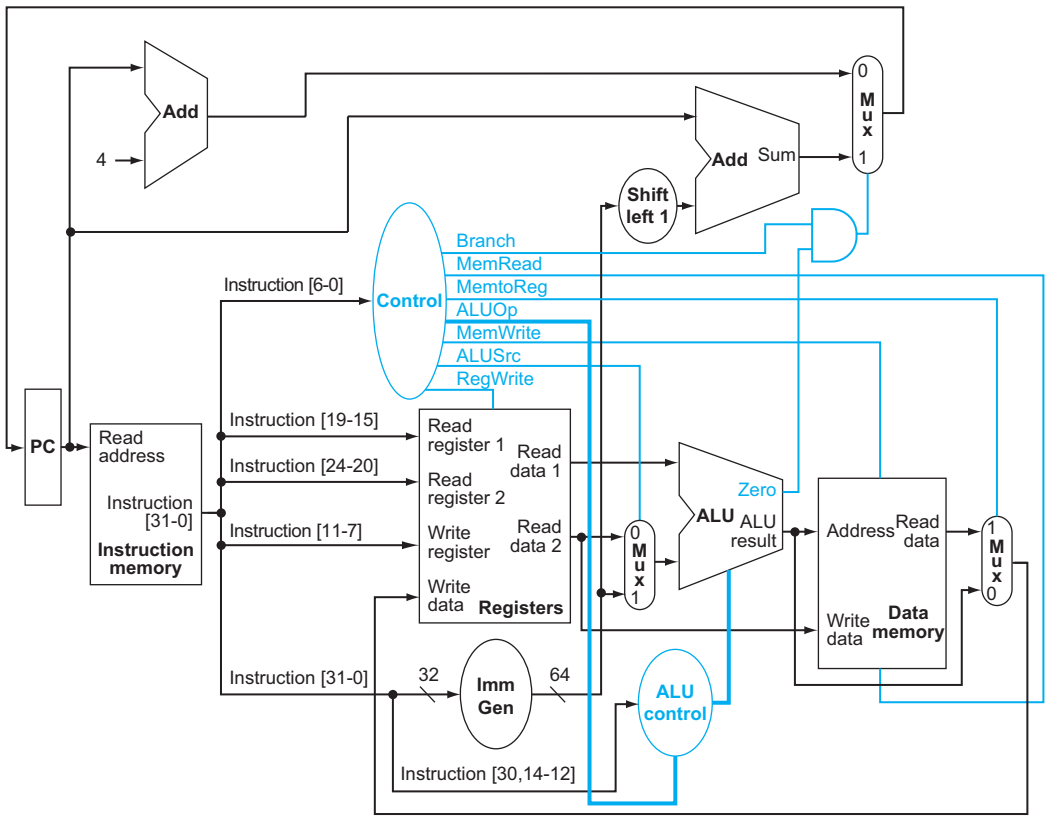
2 (b) **(Multiple Choice)** After generating an object file `foo.o` of the assembly code (`foo.s`), this object file is run through a linker with static library `lib.a`. Assuming `printf` is a function defined in `lib.a`, which of the following will be used to resolve the instruction `jal ra printf`? Circle the letter of your choice(s).

A. foo.o's symbol table
B. foo.o's relocation table
C. lib.a's symbol table
D. lib.a's relocation table

Solution: B, C

2 4. **RISC-V Control**

Choose the correct timing of the critical path for `xori` according to the data path given below.



A. $t_{\text{clk-q}} + t_{\text{IMEM}} + \max\{t_{\text{Reg}}, t_{\text{Imm}}\} + t_{\text{ALU}} + 2t_{\text{Mux}} + t_{\text{Setup}}$;

B. $t_{\text{clk-q}} + t_{\text{Add}} + t_{\text{IMEM}} + t_{\text{Reg}} + t_{\text{BComp}} + t_{\text{ALU}} + t_{\text{DMEM}} + t_{\text{Mux}} + t_{\text{Setup}}$;

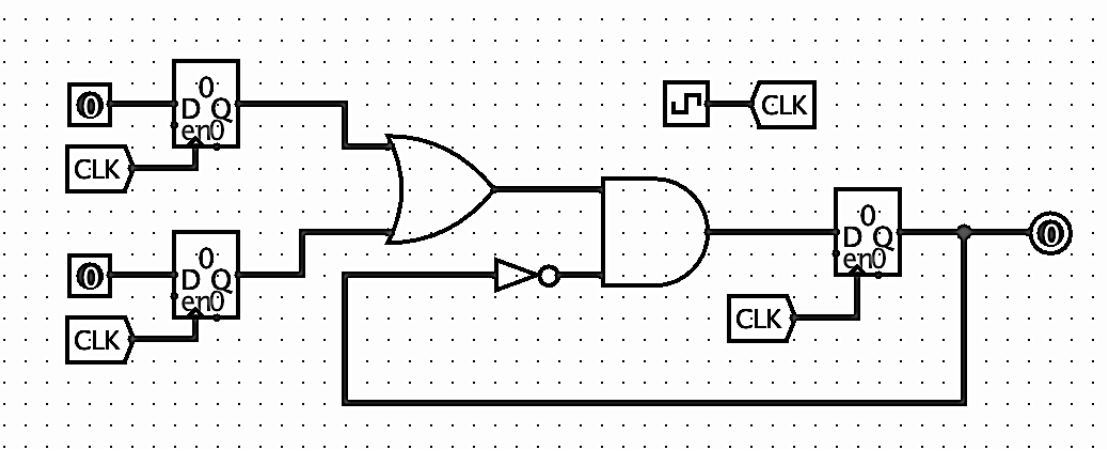
C. $t_{\text{clk-q}} + t_{\text{IMEM}} + \max\{t_{\text{Reg}}, t_{\text{Imm}}\} + t_{\text{ALU}} + t_{\text{DMEM}} + 3t_{\text{Mux}} + t_{\text{Setup}}$;

D. None of the above.

Solution: A.

5. **SDS**

Consider the following circuit below. Assume that AND and OR gates have a delay of 16 ps, NOT gates have a delay of 8 ps, and all registers have a setup time constraint of 12 ps and clock-to-Q delay of 6 ps. Assume all wires are ideal, i.e. they have zero delay



2 (a) What is the maximum possible hold time constraint for registers to function properly in this circuit?

Solution: 30ps

The shortest path between two registers goes through the NOT gate, then the AND gate, for a total delay of 16+8=24 ps.

Hold time = smallest combinatorial delay + clock-to-Q delay = 24+6 = 30 ps

2 (b) What is the minimum allowable clock period for this circuit to function properly?

Solution: 50ps.

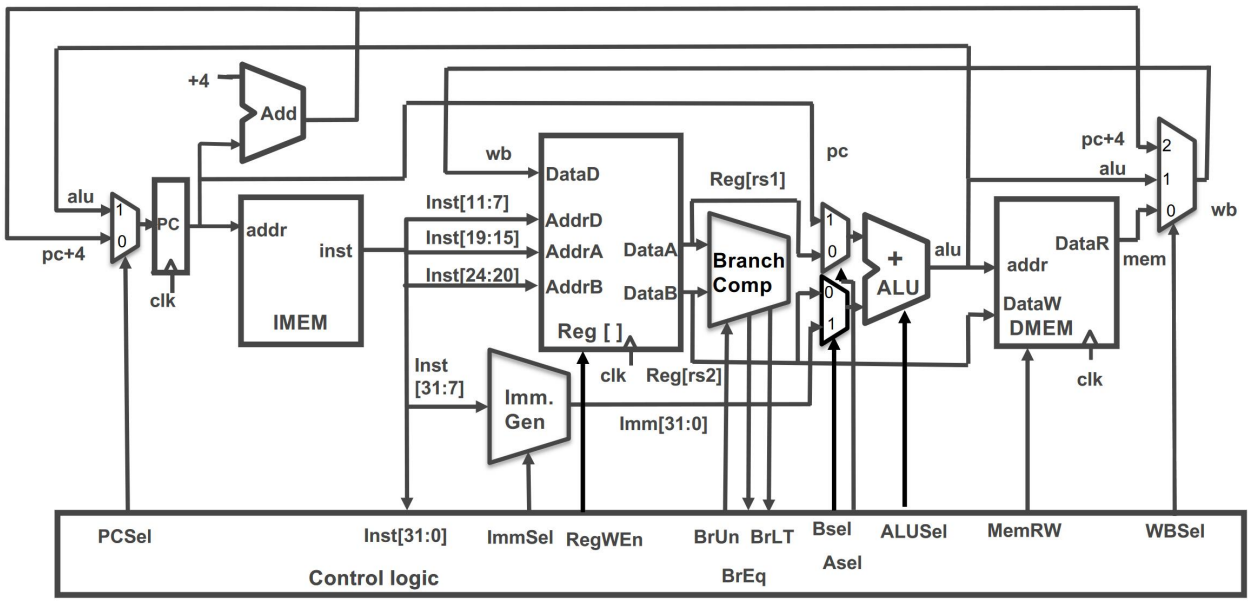
The longest path between two registers goes through the OR gate, then the AND gate, for a total delay of 16+16=32 ps.

Shortest clock period = clock-to-Q delay + largest combinatorial delay + setup time = 6+32+12 = 50 ps

2 (c) What is the maximum allowable clock frequency for this circuit to function properly, in gigahertz?

Solution: 20 GHz
1 / (50 ps) = 20 GHz

6. RISC-V Datapath



5 (a) **(Multiple Choice)** For instruction `jalr ra`, select the correct value for the control logic.

1. PCSel:

   A. alu B. pc + 4

2. ImmSel:

   A. I B. S C. B D. J E. U

3. Asel:

   A. 0 B. 1

4. Bsel:

   A. 0 B. 1

5. WBsel:

   A. pc + 4 B. alu C. mem

1	2	3	4	5

Solution: A; A; A; B; A;

2 (b) **(Multiple Choice)** The above instruction `jalr ra` is executed. Answer the following question.

1. Among the five stages of instruction execution, during which stage(s) the values in **IMEM**, **DMEM** or **Reg** is(are) changed

   A. Instruction Fetch C. Execute E. Register Write
   B. Instruction Decode D. Memory Access

2. Whose value(s) is(are) changed?

   A. IMEM B. DMEM C. Reg

1	2

Solution: E; C;

4 (c) **(True or False)**

1. For instruction `add`, the value is written back to Reg as soon as it is computed by ALU.
2. Except **Write Enable**, all the input and output buses of register file are 32-bit.
3. For register file and memory, CLK is a factor ONLY during write operation.
4. Register file holds all the registers needed for instruction execution.

1	2	3	4

Solution: F; F; T; F.

7. **Virtual Memory**

2 (a) **(Multiple choice)** Which of the following statements are True about virtual memory?

A. Page tables make it possible to store the pages of a program non-contiguously.
B. The TLB reach of a system with 4KB page sizes and 32 entries is 128KB.
C. The rate of page faults in a virtual memory system can always be reduced by adding more memory.
D. A virtual address space owned by a user cannot be larger than the physical address space.

Solution: A, B

2 (b) Assume we have 48-bit virtual addresses and 1 MiB of physical memory. The page size is 4 KiB. Answer the following questions:

(i) What is the maximum number of virtual pages that each process can have?

(ii) Assume each page could store 2^{15} Page Table Entries (PTEs). How many pages does our page table need if we use a single-level page table?

Solution: (i) 2^{36} (ii) 2^{21}

8. **Superscalar**

[8] (a) **(True or False)** Please fill your answer (T or F) in the table below.

1. The word "Superscalar" comes from "Scalar" and is used in comparison with that.
2. A superscalar processor is a CPU that implements a form of parallelism called instruction-level parallelism within a single processor.
3. In Flynn's taxonomy, a single-core superscalar processor that supports short vector operations could be classified as SISD (single instruction stream, single data stream).
4. A superscalar CPU executes multiple instructions in parallel by using multiple execution units.
5. A pipelined CPU executes multiple instructions in parallel by using multiple execution units.
6. A superscalar CPU can execute more than one process/thread at a given time.
7. A hyper-threading CPU is computing more than one process/thread at the same time.
8. A 1-instruction-wide processor in which every instruction takes a single cycle benefits from out-of-order execution because it can improve performance.

1	2	3	4	5	6	7	8

Solution: T; T; F; T; F; F; F; F.

[2] (b) Calculate the CPI (cycle per instruction) of a program with following parameters.

Operation	Freq $_i$	CPI$_i$
ALU	45%	2
Load	30%	5
Store	15%	4
Branch	25%	4

Solution: $45\% \times 2 + 30\% \times 5 + 15\% \times 4 + 25\% \times 4 = 4$ (cycles)

6 (c) Suppose there is a superscalar CPU with out-of-order execution and it has 4 functional units which execute addition and only **one** functional unit which executes division (and some other unit, which is not important).

Now it is executing the following instructions:

```
1     add     x9, x9, x9
2     div     x10, x9, x8
3     mv      x12, x6
4     add     x12, x12, x6
5     add     x11, x10, x12
6     lw      x13, 8(x12)
7     lw      x14, 8(x10)
8     mv      x7, x15
9     mv      x8, x16
10    mv      x9, x17
11    div     x7, x7, x8
12    sw      x10, 0(x12)
13    mv      x6, x7
```

Here is the state of the CPU in the 100^{th} cycle:

Reservation Station	Functional Units	Commit Unit
Waiting:	Computing:	Waiting:
5 add x11, x10, x12	2 div x10, x9, x8	3 mv x12, x6
7 lw x14, 8(x10)	6 lw x13, 8(x12)	4 add x12, x12, x6
11 div x7, x7, x8	10 mv x9, x17	8 mv x7, x15
12 sw x10, 0(x12)		9 mv x8, x16

1. Why is 5,11 still waiting?

Solution: x10 is busy. Division unit is busy.

2. Suppose 2 and 10 will be finished in this cycle(and 6 will not), what is the state of CPU in the next cycle(You can write number in the table)?

Reservation Station	Functional Units	Commit Unit
Waiting:	Computing:	Waiting:

Solution:

Reservation Station	Functional Units	Commit Unit
Waiting: 7 12	Computing: 5 6 11	Waiting: 8 9 10

9. **Cache**

2 (a) We have an 10-bit address space and a n-way set associative cache. After a while, the entire cache has the following state:

Index	Tag 1	Valid 1	Tag 2	Valid 2
0b00	0b1011	1	0b1101	1
0b01	0b0011	1	0b0010	1
0b10	0b1110	1	0b0111	0
0b11	0b1111	0	0b0001	0

Calculate the bit width of tag, index, and block offset.

Tag	Index	Offset

Solution: 4, 2, 4.

3 (b) Calculate the following parameters of the cache in (b):

Associativity: _____

Block size (in Bytes): _____

Cache size (in Bytes): _____

Solution:

1. Associativity is 2
2. Block size is 16 Bytes;
3. Cache size is 128 Bytes;

5 (c) Consider the following accesses separately. (i.e. **DO NOT update the cache state** after each memory access.) Determine if each access would be a hit or miss basing on the cache state shown in (b).

If it's a miss, classify the possible miss type(s). (If multiple miss types are possible, **select all possible miss types.**)

The answer should be Hit / Compulsory Miss / Capacity Miss / Conflict Miss or multiple of them.

`0b0011011011:` _____

`0b0010111100:` _____

`0b0011001101:` _____

`0b0110100010:` _____

`0b1110100010:` _____

Solution:

1. 0b 0011-01-1011: Hit
2. 0b 0010-11-1100: Compulsory, as the second one
3. 0b 0011-00-1101: Compulsory or Conflict Miss. Because we don't know if this block has been accessed before and ejected. This can't be a capacity miss, since there are still empty slots in our cache.
4. 0b 0110-10-0010: Compulsory. Our tag isn't in our index. This must be compulsory, because we still have an empty slot in this index. We only ever kick out a block when the cache is full, and only to replace that block with a new one; as such, we can't have kicked out a block while there is still empty space in the cache.
5. 0b 1110-10-0010: Hit

2 (d) Compute the AMAT (in ps) given the following parameters:

Cache Type	Single Level Cache
Clock period	200ps
Miss Rate	0.01
Miss penalty	60 clock cycles
Hit Time	1 clock cycle

Solution: 320

2 (e) Compute the AMAT (in ps) given the following parameters:

Cache Type	Multi-Level Cache
Clock period	100ps
L1 Miss Rate	0.02
L1 Hit Time	1 clock cycle
L2 Miss Rate	0.05
L2 Hit Time	5 clock cycles
Main Memory Hit Time	100 clock cycles

Solution: 120

3 (f) Compute the Local Miss Rate for L1 cache, Local Miss Rate for L2 cache and Global Miss Rate, Given that:

In 1000 memory accessing, 160 of them referring to L2 cache, and 40 of the 160 referring to Main memory.

Local MR for L1 cache: _____

Local MR for L2 cache: _____

Global Miss Rate: _____

Solution: 0.16, 0.25, 0.04

10. **Flynn Taxonomy, SSE**

8 (a) For a vector, we can compute its sum of squares, which equal to $\sum_{i=0}^{n-1} x_i^2$, the double vector is stored in an array whose address is aligned to a multiple of 16 bytes. Suppose n is a multiple of 2 and all the operations does not cause overflow, and double use 64-bit. Finish the following C code using the SSE intrinsics. All variables have been declared for you and you **should not** declare more variables. You might find the following intrinsics useful.(You may not use all the empty lines, just leave it empty then.)

- `__m128d _mm_setzero_pd()`
  Return vector of type `__m128d` with all elements set to zero.
- `__m128d _mm_loadu_pd (double const* mem_addr)`
  Load 128-bits (composed of 2 packed double-precision (64-bit) floating-point elements) from memory into dist.mem_addr.
- `__m128d _mm_mul_pd (__m128d a, __m128d b)`
  Multiply packed double-precision (64-bit) floating-point elements in a and b, and store the results in dst.
- `__m128d _mm_add_pd (__m128d a, __m128d b)`
  Add packed double-precision (64-bit) floating-point elements in a and b, and store the results in dst.
- `__m128d _mm_hadd_pd (__m128d a, __m128d b)`
  Horizontally add adjacent pairs of double-precision (64-bit) floating-point elements in a and b, and pack the results in dst. (i.e. `dst[63:0] := a[127:64] + a[63:0] dst[127:64] := b[127:64] + b[63:0]`)
- `double _mm_cvtsd_f64 (__m128d a)`
  Copy the lower double-precision (64-bit) floating-point element of a to dst.

```
1  double sum_of_squares(double *a, int n) {
2      __m128d result = _mm_setzero_pd();
3      __m128d tmp = _mm_setzero_pd();
4      for (int i = 0; i < n; i+=2) {
5
6          _____________________________________
7
8          _____________________________________
9
10         _____________________________________
11     }
12
13     _____________________________________
14
15     _____________________________________
16 }
```

Solution:

```
 1 double sum_of_squares(double *a, int n) {
 2     __m128d result = _mm_setzero_pd();
 3     __m128d tmp = _mm_setzero_pd();
 4     for (int i = 0; i < n; i+=2) {
 5         tmp = _mm_loadu_pd((__m128d *)(a + i));
 6         tmp = _mm_mul_pd(tmp, tmp);
 7         result = _mm_add_pd(result, tmp);
 8     }
 9     result = _mm_hadd_pd(result, result);
10     return _mm_cvtsd_f64(result);
11 }
```

11. **Various Questions**

[9] (a) **(True or False)** Fill your answer (T or F) in the table below.

1. When the processor uses polling, the control register can carry data.
2. Supervisor mode can fully isolate applications from each other or from the OS.
3. A single FPGA can be re-programmed to implement different designs at different times.
4. Functional blocks differentiate from each other mainly in terms of the output size of an LE and the number of LEs in a functional block.
5. One disadvantage of Programmed I/O is that it can have the energy cost of a general-purpose CPU, while one could use a simple hardware solution for that.
6. DMA engine contains registers which can't written by CPU.
7. The key to protocol families is that communication logically takes place at two adjacent levels of the protocol.
8. RAID-1 offers the best write performance.
9. The heartbeat can only be sent from the user to the server, not from the server to the user.

1	2	3	4	5	6	7	8	9

Solution: F; F; T; F; T; F; F; T; F.

[4] (b) **(Multiple Choice)** Fill your answer in the table below. There may be more than one correct answer.

1. Which of the following statements is true for interrupt?
   A. HDL is less tolerant of errors than PL.
   B. Both HDL and PL have the concept of synchronization to clock.
   C. HDL is more complex than PL.
   D. Time critical functions are designed in PL.
   E. Minor defects can be corrected in PL.
2. Which of the following statements is true for interrupt?
   A. Interrupt handling is done in a transparent manner.
   B. Page fault, bus error, illegal Instruction may cause an interrupt.
   C. Traps can be caused by either external or internal events.
   D. Process can enter the supervisor mode by using an trap.

1	2

Solution: ACE; ACD.