CENG 3420 Midterm (2021 Spring)

Name: _____
ID: _____

Solutions

Q0 (0 marks)

1. What is your last digit of your SID (0 is regarded as 10)? This value is defined as `NUM_1` in the whole question paper.
2. What is your last two digits of your SID (00 is regarded as 100)? This value is defined as `NUM_2` in the whole question paper.
3. What is your last three digits of your SID? This value is defined as `NUM_3` in the whole question paper.

Example: if your SID is 12345678, then $\texttt{NUM\_1} = 8, \texttt{NUM\_2} = 78, \texttt{NUM\_3} = 678$.

Q1 (10 marks)

Select and fill the correct answer.

1. RISC-V is a ____ and ____ ISA. (sol: A)

   A. little-endian RISC

   B. little-endian CISC

   C. big-endian RISC

   D. big-endian CISC

2. Virtual memory systems use ____ mechanism. (sol: A)

   A. write-back

   B. write-through

   C. write-allocate

3. The largest positive integer in IEEE754 single-precision floating-point format is ____ (sol: D)

   A. $2^{126} - 2^{103}$

   B. $2^{127} - 2^{104}$

   C. $2^{127} - 2^{103}$

   D. $2^{128} - 2^{104}$

4. ____ is a type of single address space multiprocessor in which some memory accesses are much faster than others depending on which processor asks for which word. (sol. B)

   A. UMA

   B. NUMA

5. Dividing 10101_2 by 11_2, the quotient is ____ and the reminder is ____. (sol: C)

   A. 100_2 01_2

   B. 111_2 01_2

   C. 111_2 0_2

   D. 100_2 0_2

6. ____ stores data as electric charge on a capacitor. (sol: B)

   A. SRAM

   B. DRAM

Q2 (5 marks)

1. Describe the steps that transform a program written in a high-level language such as C into a representation that is directly executed by a computer processor.
2. Write C statements that corresponds to the two RISC-V assembly instructions below.

```
srli t0, t1, 3

andi t0, t2, 0xFEF
```

   You should define variables that hold corresponding register values explicitly.

Answer:

1. The program is compiled into an assembly language program, which is then assembled into a machine language program.

2.
```
int t0;
int t1;
int t2;
t0 = t1 > > 3;
t0 = t2 & 0xFEF;
```

Q3 (10 marks)

Consider a RISC-V code snippet, and assume that the code is running on a 5-stage RISC-V core.

```
add x15, x12, x11
ld x13, 4(x15)
ld x12, 0(x2)
or x13, x15, x13
sd x13, 0(x15)
```

1. Find all data dependences in this instructioin sequence.
2. If the design has not implemented any forwarding or hazard detection, insert NOPs to ensure the correct execution.
3. If there is forwarding, for the first seven cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units.

Answer:

1. The 2nd instruction depends on the 1st instruction (x15, true data dependence); The 3rd instruction depends on the 1st instruction (x12, anti-dependence); The 4th instruction depends on the 1st instruction (x15, true data dependence); The 4th instruction depends on the 2nd instruction (x13, true data dependence & output dependence); The 5th instruction depends on the 1st instruction (x15, true data dependence); The 5th instruction depends on the 2nd instruction (x13, true data dependence); The 5th instruction depends on the 4th instruction (x13, true data dependence).
2. add x15, x12, x11
   nop
   nop
   ld x13, 4(x15)
   ld x12, 0(x2)
   nop
   or x13, x15, x13
   nop
   nop
   sd x13, 0(x15)
3. According to the pipeline graph, as shown below:

cycle	1	2	3	4	5	6	7	8	9
add	IF	ID	EX	MEM	WB				
ld		IF	ID	EX	MEM	WB			
ld			IF	ID	EX	MEM	WB		
or				IF	ID	EX	MEM	WB	
sd					IF	ID	EX	MEM	WB

   Because there are no stalls in this code, PCWrite and IF/IDWrite are always 1 and the mux before ID/EX is always set to pass the control values through.
   (1) ForwardA = XX; ForwardB = XX (no instruction in EX stage yet)
   (2) ForwardA = XX; ForwardB = XX (no instruction in EX stage yet)
   (3) ForwardA = 00; ForwardB = 00 (no forwarding; values taken from registers)
   (4) ForwardA = 10; ForwardB = 00 (base register taken from result of previous instruction)
   (5) ForwardA = 01; ForwardB = 01 (base registers taken from results of two previous instructions)
   (6) ForwardA = 00; ForwardB = 10 (rs1 = x15 taken from register; rs2 = x13 taken from

result of 1st ld – two instructions ago)
(7) ForwardA = 00; ForwardB = 10 (base register taken from register file. Data to be written taken from previous instruction)

Q4 (15 marks)

We have a loop as follows.

```
LOOP: ld a0, 0(a3)
      ld a1, 8(a3)
      add a2, a0, a1
      subi a3, a3, 16
      bnez a2, LOOP
```

Assume that we have the perfect branch prediction (i.e., no stalls due to control hazards) and there are no delay slots that the pipeline has full forwarding support, and that branches are resolved in the EX (as opposed to the ID) stage.

1. Show a pipeline execution diagram for the first two iterations of this loop.
2. Mark pipeline stages that do not perform useful work. How often while the pipeline is full do we have a cycle in which all five pipeline stages are doing useful work? (Begin with the cycle during which the `subi` is in the IF stage. End with the cycle during which the `bnez` is in the IF stage)

Answer:

1. As shown in the Figure 1

Figure 1: Pipeline execution diagram

```
ld x10, 0(x13)      IF ID EX ME | WB
ld x11, 8(x13)         IF ID EX | ME WB
add x12, x10, x11         IF ID | .. EX ME! WB
addi x13, x13, -16           IF | .. ID EX  ME! WB
bnez x12, LOOP                  | .. IF ID  EX  ME! WB!
ld x10, 0(x13)                        IF  ID  EX  ME  WB
ld x11, 8(x13)                            IF  ID  EX  ME WB
add x12, x10, x11                             IF  ID  .. EX | ME! WB
addi x13, x13, -16                                IF  .. ID | EX  ME! WB
bnez x12, LOOP                                        IF | ID  EX  ME! WB!
Completely busy                 | N  N  N   N   N   N   N  N  |
```

2. In a particular clock cycle, a pipeline stage is not doing useful work if it is stalled or if the instruction going through that stage is not doing any useful work there. As the diagram above shows, there are not any cycles during which every pipeline stage is doing useful work.

Q5 (15 marks)

Consider the following loops written in C programming language:

```
for(int i = 0; i < a; i++)
    for(int j = 0; j < b; j++)
        A[j * 4] = i + j;
```

The values of `a`, `b`, `i` and `j` are in registers `t0`, `t1`, `t2` and `t3`, respectively. The register `a0` holds the base address of the array `A`. Translate to the corresponding RISC-V assembly code.

Answer:

```
LOOPI:
addi t2, zero, 0 # init: i = 0
bge t2, t0, ENDI # while i < a
addi t4, a0, 0   # t4 = &A
addi t3, zero, 0 # init: j = 0
LOOPJ:
bge t3, t1, ENDJ # while j < b
add t5, t2, t3   # t5 = i + j
sd t5, 0(t4)     # A[j * 4] = i + j
addi t4, t4, 16  # t4 = &A[4 * (j + 1)]
addi j, j, 1     # j++
jal zero, LOOPJ
ENDJ:
addi t2, t2, 1   # i++
jal zero, LOOPI
ENDI
```

Q6 (15 marks) There is a computer that has a 64MB byte-addressable main memory. Instructions and data are stored in separated caches, each of which has eight 64B cache lines. The data cache uses **direct-mapping**. Now there are two programs in the following form.

Program A:

```
int a[64][64];
int sum_array1() {
    int i, j, sum = 0;
    for(i = 0; i < 64; i++)
        for(j = 0; j < 64; j++)
            sum += a[i][j];
    return sum;
}
```

Program B:

```
int a[64][64];
int sum_array1() {
    int i, j, sum = 0;
    for(j = 0; j < 64; j++)
        for(i = 0; i < 64; i++)
            sum += a[i][j];
    return sum;
}
```

Suppose `int` data is represented in 32-bit 2's complement and `i`, `j`, `sum` are stored in specific registers. Arrays are stored in row-major with the start address NUM_3_{10} (i.e., NUM_3 in decimal) in the main memory. Answer the following questions.

1. What are the line numbers of the main memory blocks that contain `a[0][31]` and `a[2][2]` respectively? (Cache line number starts from 0)

2. What are the data cache hit rates of program A and B?

Answer:

1. a. NUM_3 + 31 × 4 = var, $\frac{\text{var}}{64}$ = result.
   b. NUM_3 + (64 × 2 + 2) × 4 = var1, $\frac{\text{var1}}{64}$ = var2, var2 mod 8 = result
2. The size of array `a` is $64 \times 64 \times 4 = 2^{14}$B. Since the size of a block is 64B, it takes 2^8 main memory blocks to store it. Under the condition of row-major, 2^8 times of cache miss will appear. Therefore, the hit rate of program `A` is $\frac{(2^{12}-2^8)}{2^{12}} = 93.75\%$. For program `B`, the hit rate is 0.

Q7 (5 marks)

Describe two cache replacement strategies.

1. Write-through
2. Write-back

Answer:

1. It always write the data into both the cache block and the next level RAM in the memory hierarchy.
2. It writes to the memory hierarchy when the cache block is **evicted**.

Q8 (15 marks)

The following table shows the different instructions ratios

Table 1: Instruction mix

R-type	I-type (non-Id)	Load	Store	Branch	Jump
$2 \times$ NUM_1	25%	22%	8%	$45\% - 3 \times$ NUM_1	NUM_1

1. What fraction of all instructions use the data memory?
2. What fraction of all instructions use the instruction memory?
3. What fraction of all instructions use the sign extend?
4. What is the sign extend doing during cycles in whih its output is not needed?

Answer:

1. Only `Load` and `Store` use data memory. 30%
2. 100%
3. Only R-type instructions do not use the Sign extender. $1 - 2 \times$ NUM_1
4. The sign extend produces an output during every cycle. If its output is not needed, it is simply ignored.

Q9 (10 marks)

Given the following specifications of the datapath latencies:

Table 2: Specification of the datapath latencies (unit: *ps*)

Stages	IF	ID	EX	MEM	WB
Latencies (ps)	`NUM_2`	`NUM_3`	550	800	600

1. What is the clock cycle time in a pipelined and non-pipelined processor?
2. What is the total latency of an `lw` instruction in a pipelined and non-pipelined processor?
3. If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?

Answer:

1. Non-pipelined (sum of all stages): $\texttt{NUM\_2} + \texttt{NUM\_3} + 550 + 800 + 600$ *ps*; Pipelined (the slowest stage): $\texttt{max}\{\texttt{NUM\_3}, 800\}$ *ps*.
2. The instruction `lw` takes all five stages. Non-pipelined: $\texttt{NUM\_2} + \texttt{NUM\_3} + 550 + 800 + 600$ *ps*; Pipelined (takes five stages at the slowest cycle): $\texttt{max}\{\texttt{NUM\_3}, 800\} \times 5$ *ps*.
3. Split ID / MEM stage (find what is the slowest). If you split ID, and then the new clock cycle will be 800 *ps*. If you split MEM, and then the new clock cycle will be $\texttt{max}\{\texttt{NUM\_3}, 600\}$ *ps*.