Solver: Chen Zhe

Email Address: chen0892@e.ntu.edu.sg

1. (a)

   Denote the execution time for M2 is x.
   Since M1 is twice as fast as M2, execution time for M1 is x/2.
   Thus the execution time for P1 is 2/3*x/2 (two-thirds of the program runs in the speed of M1) + 1/3*x/2*1/2 (one-third of the program runs 2 times faster) = 5/12*x
   The execution time for P2 is 1/2*x*1/3 (half of the program runs 3 times faster) + 1/2*x (the other half of the program runs in the speed of M2) = 2/3*x
   So the speedup of P1 over P2 is 2/3 / (5/12) = 1.6

   (b)
   (i)
   There are data hazard and control hazard in the code segments. (it's arguable to include structural hazard as well, but basic MIPS processors that we've learnt will have 2 RAMs to avoid it, and register file natively supports read and write during the same clock cycle)
   Data dependencies (although not specifically written, the "dependency" referred in the question can only be data dependency in the context of this course) in the code segments are RAW (read after write) and WAR (write after read).

   (ii)
   (It is a bit confusing that this question asks for dependency again, but I just proceed to draw the pipelined execution diagram and indicate type of dependency in the diagram. Note that data forwarding is allowed here)

I1	IF	ID	EXE	MEM	WB							
I2		IF	ID	EXE	MEM	WB						
I3			IF	ID	EXE	MEM	WB					
I4				Stall	IF	ID	EXE	MEM	WB			
I5						IF	ID	EXE	MEM	WB		
I6							IF	ID	EXE	MEM	WB	
I7								IF	ID	EXE	MEM	WB

   The arrows drawn shows the forwarding path.
   Assume that BNEZ instruction makes use of ALU's zero flag to decide whether branching should happen. 2 stall cycles should be inserted after the branching instruction to mitigate control hazard. (the question is not clear as to what is the number they want us to answer, so make appropriate assumptions and present all you have for the question)
   Therefore, as the loop will execute for 100 times, the total number of stall cycles are 300 during code execution.

   (iii)
   Program counter is updated with branch target address at writeback stage means next Instruction Fetch should happen after BNEZ instruction's Write Back stage, assuming Instruction Memory is clocked. 5 stalls are necessary in this case. Therefore, the total number of clock cycles required is 1 loop initialization + (6 instructions + 6 stalls) * 100 times = 1201 cycles

If there are errors, please report using the form in bit.ly/SCSEPYPError

(c)

Since the maximum operating frequency varies linearly with operating voltage, the maximum frequency that the processor can run under 3V is 400MHz/4V*3V = 300MHz.

Processor run under 4V:

P(4Vdynamic) = A*C*4^2*400MHz

P(4Vstatic) = 4*I(leak)

Processor run under 3V:

P(3Vdynamic) = A*C*3^2*300MHz

P(3Vstatic) = 3*I(leak)

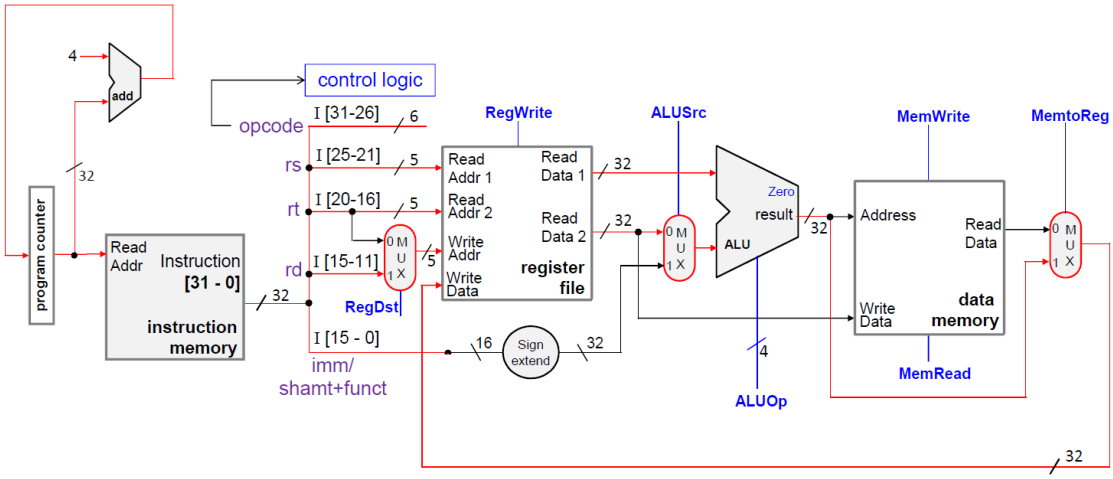
Note: leak current increases with temperature. Assume the processor is cooled to the same temperature in both cases

% change in dynamic power = (P(4Vdynamic)-P(3Vdynamic)) / P(4Vdynamic) = 57.81%

% change in static power = (P(4Vstatic)-P(3Vstatic)) / P(4Vstatic) = 25%

2. (a)

From Lecture Slide, removed ALU control and ALUOp comes directly from the control logic:



First the address of this ADD instruction will be in the program counter. At the next clock cycle, the instruction memory will output the instruction content.

The 6-bit op-code that represents ADD instruction is fed into control logic and it outputs the control signals. (RegWrite is set to 1 to allow register writeback, RegDest is set to 1 since ADD is a R-type instruction. ALUSrc is set to 0 since ADD is a R-type instruction. ALUOp is set to Add mode. MemWrite and MemRead are set to 0 since ADD does not operate on date memory. MemtoReg is set to 1 to pass through ALU result to Register File's WriteData port.)

The register file also reads out the register content of register #8 and #10 respectively. The destination register number #12 is fed to the WriteAddr port.

ALU takes the 2 inputs, add them up and output to the result port.

Since the data memory is disabled, it will not do anything. The ALU result will go through MemtoReg MUX and connect to the Register File's WriteData port, which will then be written to register #12.

If there are errors, please report using the form in bit.ly/SCSEPYPError

(b)

The 2 common register file ports that ADD and ADDI use are Read Addr 1 and Write Addr. So, we can modify the instruction format of ADD to be

6-bit Opcode	5-bit Source Register #1	5-bit Destination Register	5-bit Source Register #2	Other bits

ADDI's instruction format remains unchanged:

6-bit Opcode	5-bit Source Register	5-bit Destination Register	16-bit Immediate Value	Other bits

The 5-bit source register #2 is connected to the Read Addr 2 port. Although it is affected by the first 5 bits of the immediate value in I-type instructions, since the Read Data 2 port is not used in the I-type instruction, the result is not affected in any way.

(c)

One of the solutions:

```
#set up stack to store return address
ADDI $sp, $sp, -4
SW $ra, 4($sp)
#obtain array B sum
ADDI $a0, $s0, 0
JAL array_sum
SRA $a1, $v0, 6
#store array B average in register t1
ADDI $t1, $a1, 0
#obtain array C sum
ADDI $a0, $s1, 0
JAL array_sum
SRA $a2, $v0, 6
#store array C average in register t2
ADDI $t2, $a2, 0
#compare array B and C average
JAL max-2
ADDI $a1, $v1, 0
#obtain array D sum
ADDI $a0, $s2, 0
JAL array_sum
SRA $a2, $v0, 6
#store array D average in register t3
ADDI $t3, $a2, 0
#compare array D average
JAL max-2
#store the maximum average into memory
LUI $t4, 0xA345
ORI $t4, $t4, 0xB234
SW $v1, 0($t4)
#return
LW $ra, 4($sp)
```


If there are errors, please report using the form in bit.ly/SCSEPYPError

```
ADDI $sp, $sp, 4
JR $ra
```

3. (a)

(i)

Assume data forwarding is not allowed and branching instruction takes 2 NOP cycles to mitigate control hazard.

Way-1	Way-2
I1	I5
NOP	NOP
NOP	NOP
I2	NOP
I4	I6
NOP	NOP
I3	NOP

CPI = 7 clock cycles / 6 instructions = 1.167

(ii)

Loop unrolled by 2:

I1 Loop:	LW $t0, 0($s1)
I2	LW $t1, 4($s1)
I3	ADDI $t0, $t0, 15
I4	ADDI $t1, $t1, 15
I5	SW $t0, 0($s1)
I6	SW $t1, 4($s1)
I7	ADDI $s1, $s1, 8
I8	ADDI $s2, $s2, -8
I9	BNEZ $s2, Loop

Scheduling in the 2-way superscalar machine:

Way-1	Way-2
I1	I2
NOP	I8
NOP	NOP
I3	I4
I7	I9
NOP	NOP
I5	I6

CPI = 7 clock cycles / 9 instructions = 0.778

The cycles are used more efficiently and the CPI is lowered to below 1.

(b)

Assume a single-core processor runs at frequency F under voltage V. To achieve the same performance, a dual-core processor only needs to run at frequency F/2 for each core. Since $f_{max} \propto (V - V_{th})^2 \,/\, V$, and Vth is negligible, each core only needs to run under voltage V/2.

Dynamic power of single core processor: $A \times C \times V^2 \times F$

Dynamic power of dual-core processor: $2 \times A \times C \times (\frac{V}{2})^2 \times \frac{F}{2} = \frac{1}{4} \times A \times C \times V^2 \times F$

If there are errors, please report using the form in bit.ly/SCSEPYPError

It is evident that a dual-core processor only consumes a quarter of the dynamic power to achieve the same performance as a single-core processor, which is the application of voltage-frequency scaling.

(c)
SIMD process takes 2N+1 instructions to execute an N by N matrix multiplied by N element array. In this case, 5 instructions are needed.

$$Ab = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

INSTRUCTION-1 INSTRUCTION-2

$t_{00} = a_{00}\, b_0$ $t_{01} = a_{01}\, b_1$
$t_{10} = a_{10}\, b_0$ $t_{11} = a_{11}\, b_1$

INSTRUCTION-3 Initialize $c_0 = 0$, $c_1 = 0$

INSTRUCTION-4 INSTRUCTION-5

$c_0 = c_0 + t_{00}$ $c_0 = c_0 + t_{01}$
$c_1 = c_1 + t_{01}$ $c_1 = c_1 + t_{11}$

GPP takes nearly 3N^2 instructions to do the same, in this case 10 instructions (execute the same instructions as SIMD, but line by line sequentially)

4. (a)
Assume L1 cache miss rate is **x**.
2.4 = 1 + 80*x
x = 1.75%
"improve AMAT by 65%" in the question means to achieve an AMAT of 2.4*65% = 1.56 cycles
Assume the hit rate of L2 cache is **a**.
Thus, 1.56 = 1 + 1.75% * (6 + (1-a) * 80)
Solve for a = 67.5%
(I have to admit that this is a strangely high number)

(b)
(i)
64MB byte-addressable physical memory → 26 bit physical address
8KB page size → 13 bit offset
Thus, physical page number takes 13 bits and virtual page number takes 19 bits.
Page table is indexed by virtual pages, therefore there are 2^19 entries in the single-level page table.
Each entry has to store the 13-bit physical page number and 3 information bits, totaling 16 bits, which is 2 bytes.
Thus the size of a single-level page table is 2^20 bytes = 1MB

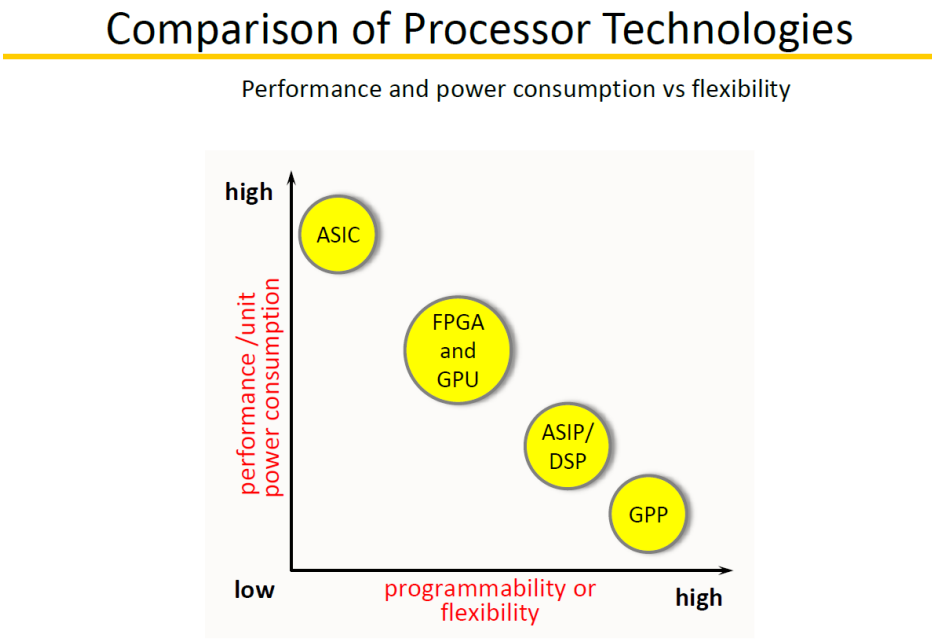
(ii)
For 0x00001524:
0x00001 = 0000 0000 0000 0000 000 1 in binary

If there are errors, please report using the form in bit.ly/SCSEPYPError

The first 19 bits are all 0 → virtual page number is 0
Virtual page #0 is valid and is mapped to physical page #8.
8 in hex translates to 00 0000 0001 000 (13-bit physical page number)
Replace the virtual page number with physical page number, we get the physical memory address: 0x0011524 (26 bits, truncate MSB)

For 0x00008FFF:
0x00008 = 0000 0000 0000 0000 100 0 in binary
The first 19 bits is 4 in decimal.
Virtual page #4 is not valid and a page fault occurs. It can't be translated to physical memory now.

(c)



Source: Michael J. Flynn, Wayne Luk, Computer System Design: System-on-Chip, John Wiley & Sons, 2011

From lecture slide, explain if you still have time.
Note that ASIC is Application Specific Integrated Circuit while ASIP is Application Specific Instruction-set Processor. These two are different in terms of power consumption and flexibility.

For reporting of errors and errata, please visit pypdiscuss.appspot.com
Thank you and all the best for your exams! ☺