Yu xinmiao 于欣淼 518021910792

VE370 Homework 1

Q1

$$\textbf{CPU Time} = \frac{\text{IC} \times \text{CPI}}{\text{Clock Rate}} = \frac{100000 \times 1.5}{500\text{MHz}} = \frac{IC_B \times 2.3}{850\text{MHz}}$$

Then from the equation, calculate that $IC_B \approx 110870$.

Q2

$$\textbf{CPU Time} = \frac{\text{IC} \times \text{CPI}}{\text{Clock Rate}} = \frac{1100 \times 1 + (200 + 150) \times 7 + 120 \times 3}{2 \times 10^9} = 1.955 \times 10^{-6}\ [s]$$

$$\textbf{CPI} = \frac{\text{CPU time} \times \text{Clock Rate}}{IC} = \frac{1955 \times 2}{1570} \approx 2.49$$

Q3

1)

$$\textbf{CPU Time} = \frac{\text{IC} \times \text{CPI}}{\text{Clock Rate}} = \frac{2560 \times 1 + (1280 \times 4) + (256 \times 2)}{3 \times 10^9} = 2.73 \times 10^{-}6\ [s]$$

2) $$\textbf{CPU Time} = \frac{1280 \times 1 + (640 \times 6) + (128 \times 2)}{3 \times 10^9} \approx 1.79 \times 10^{-6}\ [s]$$

3) $$\textbf{CPU Time} = \frac{640 \times 1 + (320 \times 8) + (64 \times 2)}{3 \times 10^9} \approx 1.11 \times 10^{-6}\ [s]$$

4) $$\textbf{CPU Time} = \frac{320 \times 1 + (160 \times 10) + (32 \times 2)}{24 \times 10^9} \approx 6.61 \times 10^{-7}\ [s]$$

Q4

x = x - y + z -72

```
add   $t0, $s0, $s2   # $t0 = x + z
addi  $t1, $s2, 72    # $t1 = y + 72
sub   $s0, $t0, $t1   # $s0 = $t0 - $t1
```

Q5

B[8] = i + A[j]

```
sll   $t2, $t1, 2     # $t2 = $t1 * 4 = 4j
add   $t2, $t2, $s5   # $t2 = address of A[j]
lw    $t3, 0($t2)     # load from memory to register, $t3 = A[j]
add   $t3, $t3, $t0   # $t3 = i + A[j]
sw    $t3, 32($s6)    # store the result from the register to memory
```

Q6

Assume `a, b, c, d` in `$s0, $s1, $s2, $s3` respectively. The address of array A is 0x00000100, such that A[0] = 0x000011f0

```
/*** Line by line ***/  // e:0x
temp0 = &A[1]           // temp0: 0x00000104  *temp0: 0x0000F1a4
temp1 = &A[0]           // temp1: 0x00000100  *temp1: 0x000011f0
A[2] = temp1            // *0x00000108 = 0x00000100
temp0 = A[1]            // temp0: 0x0000F1a4
a = temp0 + temp1       // 0x0000F1a4 + 0x00000100
/***Simplify***/
a = A[1] + &A[0]
```

The value of $s0 is 0x0000F2a4

Q7

```
lui   $s0, 0x1000   # $s0 = 0x10000000
lb    $s2, 2($s0)
```

In `$s2` : 0x00000066

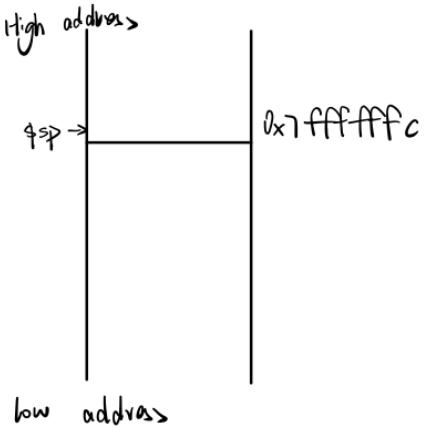
Q8

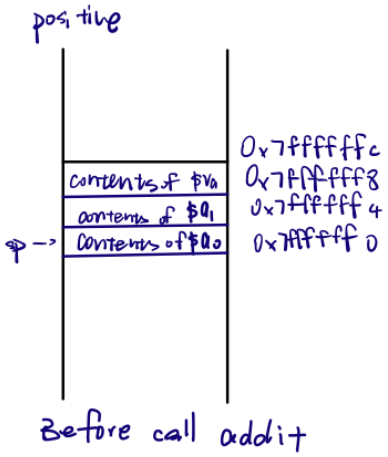
```
slt $t2, $t0, $t1       # signed $t0 < $t1, so $t2 = 1
beq $t2, $0, ELSE       # $t2 != 0, not branch to ELSE
j DONE                  # jump to done
ELSE: addi $t2, $0, -2  # not executed
DONE: ......
```

The value of `$t2` is 1.

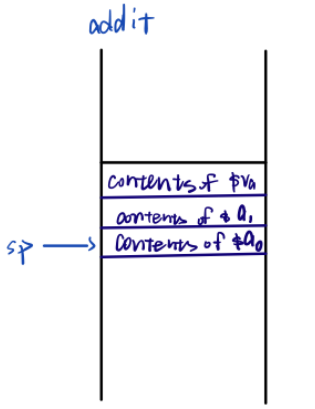
Q9

```
positive:
    addi $sp, $sp, -12  # adjust stack to make room for 3 items
    sw $ra, 8($sp)      # store the return address
    sw $a1, 4($sp)      # store the function arguments
    sw $a0, 0($sp)
    jal addit
    lw $a0, 0($sp)      # restore function arguments
    lw $a1, 4($sp)
    lw $ra, 8($sp)      # restore return address
    addi $sp, $sp, 12   # pop 3 items from stack
    slt $v0, $zero, $v0 # if (0 < $v0), $v0 = 1; else $v0 = 0
    jr $ra
addit:
    add $t0, $a0, $a1   # parameter variable a, b correspond to the
argument register
    add $v0, $t0, $zero # copy into a return register
    jr $ra
```

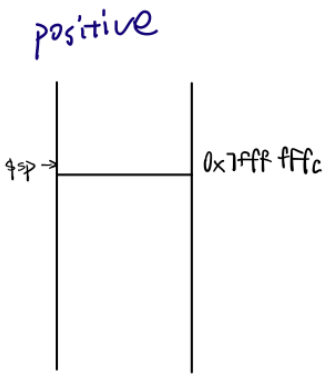




Before call addit



no change in stack when call addit



In positive, before jr $ra

Yu xinmiao 于欣淼 518021910792

VE370 Homework 2

Q1

refer to the figure

```
lui   $s0, 0x0F00   # $s0 = 0x0F000000
lbu   $s1, 2($s0)
```

The content in $s1 is 0x00000047

Q2

```
FACT:
    addi    $sp, $sp, -8    ## should be -8
    sw      $ra, 4($sp)
    sw      $a0, 0($sp)
    add     $s0, $0, $a0
    slti    $t0, $a0, 2
    beq     $t0, $0, L1
    addi    $v0, $zero, 1
    # mul $v0, $s0, $v0
    addi    $sp, $sp, 8
    jr      $ra
L1:
    addi    $a0, $a0, -1
    jal     FACT
    # addi $v0, $0, 1
    lw      $a0, 0($sp)       ## the offset do not meet the offset
where stores
    lw      $ra, 4($sp)
    addi    $sp, $sp, 8       ## should be 8
    mul     $v0, $s0, $v0
    jr      $ra
```

Q3

in binary, opcode and func, indicate the subtraction

opcode	rs	rt	rd	shamt	funct
000000	01011 $t3	01010 $t2	01001 $t1	00000	100010 (0x22)

```
sub   $t1, $t3, $t2
```

2. R-type

Q4

`lw $s1, -32($s2)`

1. 1000 1110 0101 0001 1111 1111 1110 0000

opcode	rs	rt	Immediate
100011	10010 $s2	10001 $s1	1111 1111 1110 0000

2. I-type

Q5

1. To represent the address of register, we need 7 bits ($2^7 = 128$). The bits needed for `opcode` and `funct` should not change. So **38 bits in total**.

opcode	rs	rt	rd	shamt	funct
6 bits	7 bits	7 bits	7 bits	5 bits	6 bits

2. **36 bits in total**

opcode	rs	rt	Immediate
6 bits	7 bits	7 bits	16 bits

Q6

1. For registers, 6 bits needed ($2^6 = 64$). `beq` is an I-type instruction

opcode	rs	rt	Immediate
6 bits	6 bits	6 bits	14 bits

only 14 bits could be used to represent the relative address and calculate the target address.

2. `jr` is a R-type instruciton.

opcode	rs	rt	rd	shamt	funct
6 bits	6 bits	6 bits	6 bits	2 bits	6 bits

As Pc = R[rs], the range of address do not be impacted.

Q7

```
LOOP:
    slt  $t5, $zero, $t0
    bne  $t5, $zero, ELSE
    j    DONE
ELSE:
    addi $s3, $s3, 2
    addiu $t2, $t2, 1
DONE:
    j LOOP ...
```

Convert into machine code (binary)

address						
0x1000F400 **LOOP:**	000000	00000	01000	01101	00000	101010
0x1000F404	000101	01101	00000	0000 0000 0000 0001		
0x1000F408	000010	0000 0000 0000 1111 0100 0001 01				
0x1000F40c **ELSE:**	001000	10011	10011	0000 0000 0000 0010		
0x1000F410	001001	01010	01010	0000 0000 0000 0001		
0x1000F414 **DONE:**	000010	0000 0000 0000 1111 0100 0000 00				

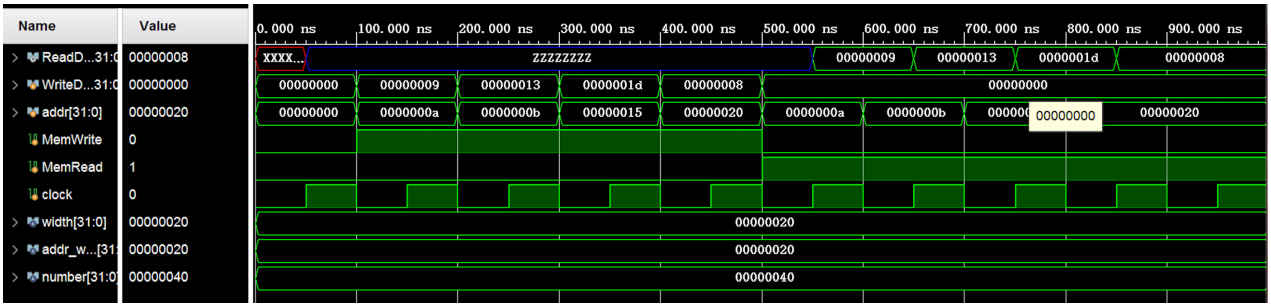
Q8

```
// memory block
module memory(ReadData, WriteData, addr, MemWrite, MemRead, clock);
  parameter width = 32;
  parameter addr_width = 32;
  parameter number = 2**addr_width;
  output [width-1:0] ReadData;
  input [width-1:0] WriteData;
  input [addr_width-1:0] addr;
  input MemWrite, MemRead, clock;
  reg [width-1:0] ReadData;
  reg [width-1:0] memory [number-1:0];
  always @(posedge clock)
  begin
    ReadData = 'bz;
    if (MemWrite) memory[addr] = WriteData; // if MemWrite and
MemRead is 1 at the smae time
    else if (MemRead) ReadData = memory[addr];  // only MemWrite
will be valid
  end
endmodule
```

```
// testbench
module testmemory;
  parameter width = 32;
```

```
  parameter addr_width = 32;
  parameter number = 2**6; // for a faster simulation
  wire [width-1:0] ReadData;
  reg [width-1:0] WriteData;
  reg [addr_width-1:0] addr;
  reg MemWrite, MemRead, clock;
  memory #(width, addr_width, number) UUT (ReadData, WriteData,
addr, MemWrite, MemRead,
  clock);
  initial begin
        #0      clock = 0; MemRead = 0; MemWrite = 0; WriteData = 0;
addr = 0;
        #100    MemWrite = 1; WriteData = 9; addr = 10;
        #100    WriteData = 19; addr = 11;
        #100    WriteData = 29; addr = 21;
        #100    WriteData = 8; addr = 32;
        #100    MemRead = 1; MemWrite = 0; WriteData = 0; addr = 10;
        #100    addr = 11;
        #100    addr = 21;
        #100    addr = 32;
  end
  always  #50 clock = ~clock;
  initial #2000 $stop;
endmodule
```

The simulation result is



Q9

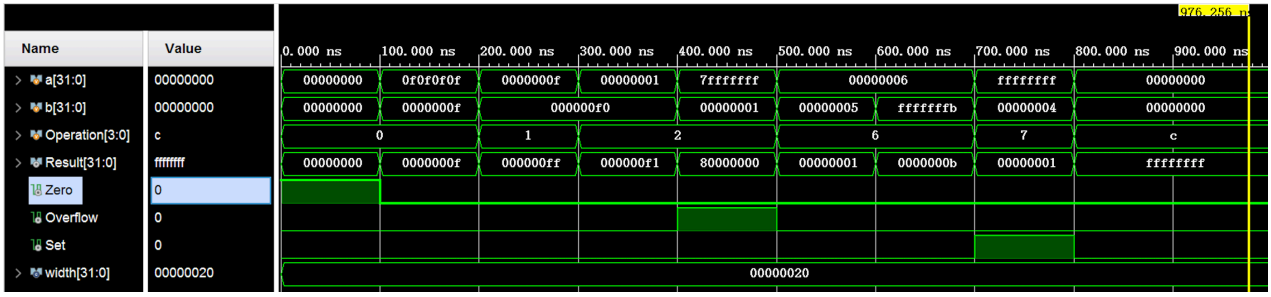
```
// ALU
module ALU#(parameter width = 32)(result, zero, overflow, set, a, b,
operation);
    output zero, overflow, set;
    output [width-1:0] result;
```

```
    input [width-1:0] a, b;
    input [3:0] operation;
    reg [width-1:0] result;
    reg zero, overflow, set;

    always@(*)
    begin
    overflow = 0;
    zero = 0;
    set = 0;
    case (operation)
    4'b0000: result = a & b;
    4'b0001: result = a | b;
    4'b0010:
        begin
        result = a + b;
        overflow = a[31] & b[31] & (~result[31]) | (~a[31]) &
(~b[31]) & result[31];
        end // test overflow based on whether different sign bits
presented
    4'b0110:
        begin
        result = a - b;
        overflow = a[31] & (~b[31]) & (~result[31]) | (~a[31]) &
(b[31]) & result[31];
        end
    4'b0111: begin // for 2's complement
        if (a[31] != b[31]) begin
          if (a[31] > b[31]) begin
            result = 1;
          end else begin
            result = 0;
          end
        end else begin
          if (a < b)
          begin
            result = 1;
          end
          else
          begin
            result = 0;
          end
        end
```

```
      set = result[31];
      end
    4'b1100: result = ~(a|b);
    endcase
    if (result == 0) zero = 1;
    end
endmodule
```

```
module ALUtest;
  parameter width=32;
  reg [width-1:0] a, b;
  reg [3:0] operation;
  wire [width-1:0] result;
  wire zero, overflow, set;
  ALU #(width) UUT (result, zero, overflow, set, a, b, operation);
  initial
    begin
    #0  a = 0; b = 0; operation = 0; // And -> result = 0, zero = 1
    #100 a = 32'b00001111000011110000111100001111; b = 15; operation
= 4'b0000; // And -> result = 15 = 0xf
    #100 a = 15; b = 240; operation = 4'b0001; // Or -> result = 255
= 0xff
    #100 a = 1; b = 240; operation = 4'b0010; // Add -> result = 241
= 0xf1, overflow = 0
    #100 a = 32'b01111111111111111111111111111111; b = 1; // Add ->
result = 0, Overflow = 1
    #100 a = 6; b = 5; operation = 4'b0110; // Sub -> result = 1
    #100 a = 6; b = -5; // result = 11
    #100 a = 32'b11111111111111111111111111111111; b = 4; operation
= 4'b0111; // Slt -> set = 1
    #100 a = 0; b = 0; operation = 4'b1100; // Nor -> result = 1
    end
  initial
  #2000 $stop;
endmodule
```



Q10

- In procedure A, `lui` is load the address of `X` ; `jr` need to jump to the address in the register `$ra`
- In procedure B, with the 2's complement, 0x10008000($gp) + 0xFFFF8040(offset) = 0x10000040(Y).

  because address `0` in B means 0x00400140, then the address `0x180` is 0x00400140+0x180= 0x004002c0. Then when `jal (address of A)` in B, `$ra` stores the $PC + 4$, which is 0x004002c4

Executable file header			
	Text size	0x440	
	Data size	0x90	
Text segment	Address	Instruction	
	0x0040 0000	lui $at, 0x1000	001111 00000 00001 0001000000000000
	0x0040 0004	ori $a0, $at, 0x1000	001101 00001 00100 0001000000000000
	...		
	0x0040 0084	jr $ra	000000 11111 00000 00000 00000 001000
	...		
	0x0040	sw $a0,	101011 11100 00100 1000 0000 0100

	0140	8040($gp)	0000
	0x0040 0144	jmp 0x040 02c0	000010 0000 0100 0000 0000 0010 1100 00
	...		
	0x0040 02c0	jal 0x0400000	000011 0000 0100 0000 0000 0000 0000 00
	...	...	
Data segment	Address		
	0x1000 0000	(X)	
	...	...	
	0x1000 0040	(Y)	

VE370 Homework 3

Q1

```
slti Rt, Rs, Imm
```

a. ALUSrc: **1** b. RegDst: **0** c. ALUOp: **10**
d. MemWrite: **0** e. MemRead: **0** f. Branch: **0**
g. MemtoReg: **0** h. RegWrite:**1**

Q2

```
andi Rt, Rs, Imm
```

f. Adder for Branch target calculation

Q3

```
SW Rt, Offs(Rs)
```

b. Data Memory

Q4

For instruction that needs to load, such that `lw`. After reading instruction from *(1)instruction memory*,the *(2)register file*. Meanwhile, *(2)Control* could executed. *(3)MUX and ALU* exectuted, then *(4)Data memory and MUX* executed. Finally, write back to *(5)Register file*. *Sign Extend* could also be executed while *(2)register file*.

it needs $(1)400 + (2)180 + (3)(80 + 150) + (4)(320 + 80) + (5)180 = 1390[ps]$ time. Then the maximum clock frequency is determined by the instruction that need most time, so it is

$Feq_{max} = \frac{1}{1390[ps]} = 7.19 \times 10^8[Hz]$.

The instruction with longest latency is lw.

For lw, the critical path in this question is Inst => Reg => ALU => Data =>Mux 1170ps (For the first input of ALU, it comes from Inst => Reg = 400+**18**0; for the second, it comes from max{Inst => Control => MUX = 400 +100+**8**0, Inst => SignExt => MUX = 400+**5**0+**8**0} = Inst => **Control** => MUX = 400 +**10**0+**8**0, and the control sign comes from Inst => Control => ALU Control = 400+**8**0 +**8**0.

Q5

Design a test that would have different value if bit 16 of Instruction Memory is stuck at 0. Set the vaule in register in $t0, $t1 as 0, 1 respectively. Use instruction

```
add $t0, $t1, $zero
```

If the value in $t0 is 0, then a stuck-at-0 fault on bit 16 of ouput of the Instruction Memory happens, other wise if value in $t0 is 1, no such problem.

Because for R-type instruction, the rt is set on bit 20-16, where $t1 has 01001 but will output 01000 when fault exists. After the execution, the value in $t0 is supposed to change from 0 to 1, if it has a stuck-at-0 fault on bit 16 of ouput of the Instruction Memory, $t0 = $t0 + 0 the value in $t0 will remain at 0.

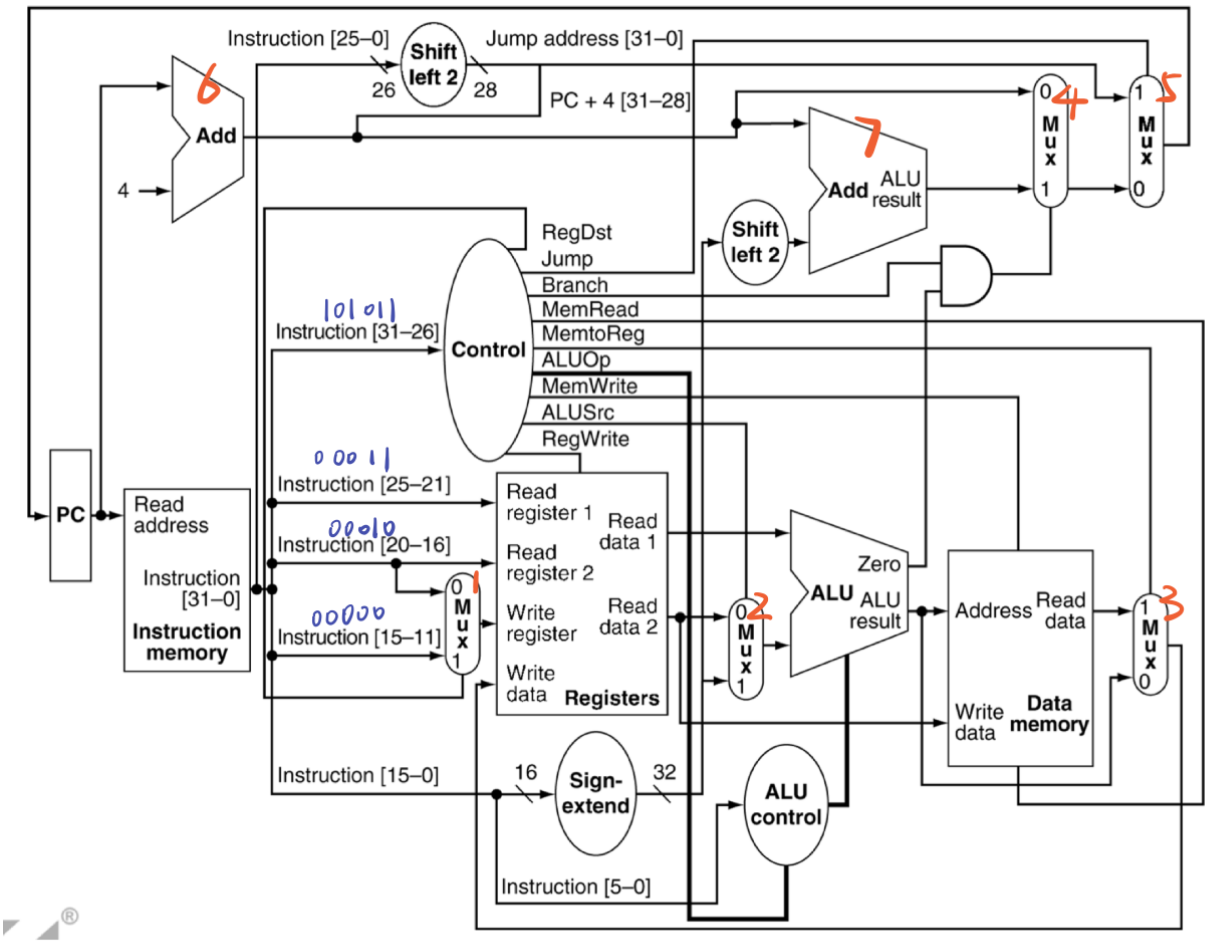
Q6

`101011 00011 00010 0000000000010100` From opcode, it is I-type instruction `sw`

opcode	Rs	Rt	Imm
101011 0x2b `sw`	00011 `$3`	00010 `$2`	0000000000010100 (20_{10})

`sw $2 (20)$3`.

R[$2] = -2; R[$3] =3; M[3 + 20]=M[23], so the intruction is to save M[23]=-2

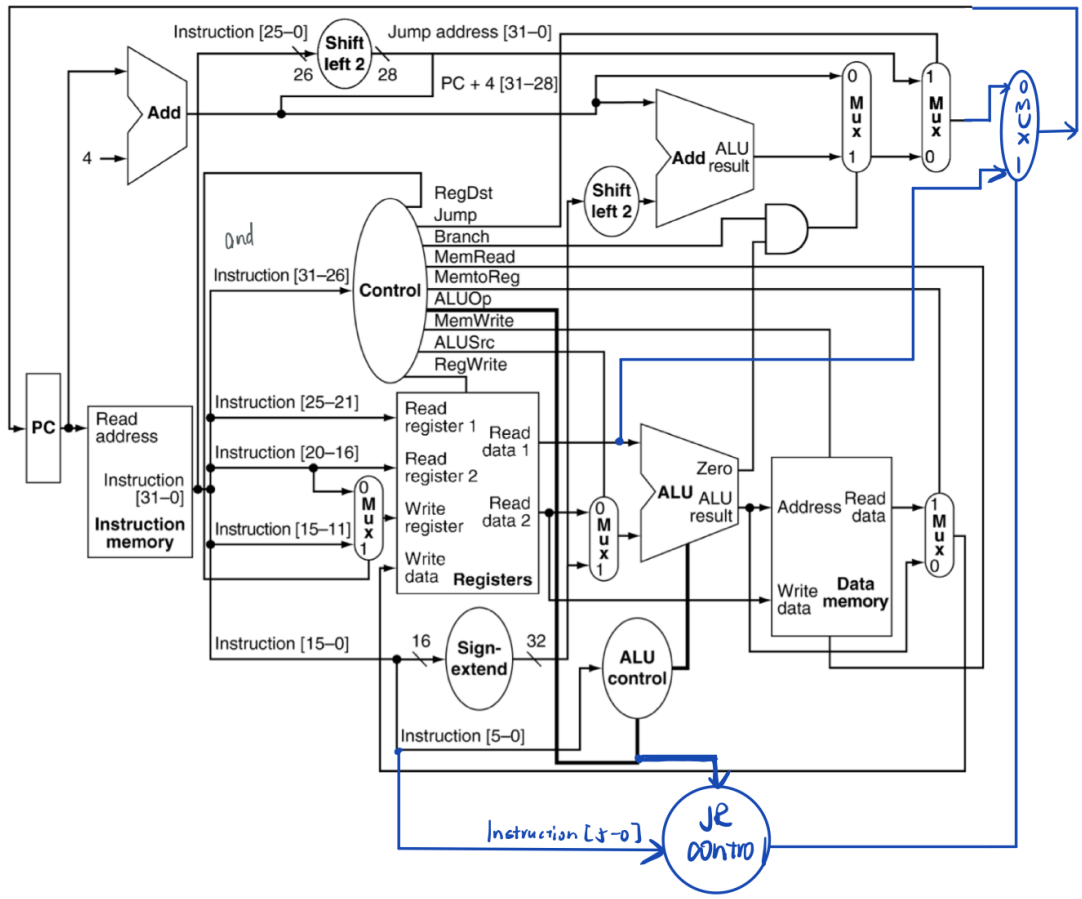


1. MUX
   - For the **MUX(1) to choose Write register**, the output of this MUX will have no effect as no writing data back to the register (`RegDst=X`). Let's set `RegDst=0`, then the output will be 00010.
   - For the **MUX(2) to choose which data is used for ALU**. output $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001\ 0100 = 20_{10}$, which is the offest with sign extension, used to calculate address (`ALUSrc=1`).
   - For the **MUX(3) to choose which data to write back** , `MemtoReg=X`, let's set it as 0, output $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001\ 0111$,
   - For the **MUX(4)** output is $PC + 4$, as `Branch=0`.
   - For the **MUX(5)** output is $PC + 4$, as `Jump=0`.
2. ALU and ADD
   - For the **add unit(6)** to calculate next instruction address, the input is the current PC and 4.
   - For the **add unit(7)** to calculate next instruction address for brach, the input is the $PC + 4$ and
     $80_{10} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 01010000$, from sign extension

shift left by 2.
    - For the **ALU**, input: from read data 1is $3_{10} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011$ and from sign-extension $20_{10} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001\ 0100$.
3. Register unit
    - **Read Register 1:** 00011
    - **Read Register 2:** 00010
    - **Write Register:** 00010, set `RegDst=0`
    - **Write Data**: X, $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001\ 0111$, as set `MemtoReg=0`, will be the output of MUX(3), the address of data memory to be written
    - **RegWrite:** 0

Q7

For `jr`, Instruction[31:26]=0 and Instruction[5:0]=0x08, need to set $PC = R[rs]$.



The logic for JR control is

ALUOp	Instruction[5:0]	Output
00	00100	1
Otherwise	Otherwise	0

Q8

First identify that without control unit's latency time, the most critical instruction is the load instruction, which has **Instr. Mem, Reg File, Mux, ALU, Data Mem, Mux and Reg File** along the path.

1. 250[ps].

   Before `MemWrite` is used, **Data Memory** needs Address and Write data. After **Instr. Mem**, Control get its input and could generate `MemWrite`. The critical path is $200 + 100 + 30 + 120 + 300 + 30 + 100 = 880[ps]$, which decides the clock cycle time. As `MemWrite` is a write enable signal for the **Data Mem**. So the time is $880 - 200 - 300 - 30 - 100 = 250[ps]$.

2. `ALUOp`, 80[ps].

   Analyze each control signal generated by **Control Unit**.

   - `RegWrite` : before one **Reg File**, $880 - 200 - 100 = 580[ps]$ at most.
   - `Jump` need to finish before one **Mux** before next rising edge, $880 - 200 - 30 = 650[ps]$ at most.
   - `RegDst` and `MemtoReg` need to finish before one **Mux and Reg File** before next rising edge, $880 - 200 - 30 - 100 = 550[ps]$ at most.
   - `Branch` needs to be generated before two **Mux** before next rising edge, $880 - 200 - 30 - 30 = 620[ps]$ at most.
   - `MemRead` is needed before **Data Mem**, one **Mux** and **Reg File**, $880 - 200 - 300 - 30 - 100 = 150[ps]$ at most.
   - `ALUOp` is needed before **ALU Ctrl, ALU** , **Data Mem** and one **Mux, RegFile**, $880 - 200 - 50 - 120 - 300 - 30 - 100 = 80[ps]$.

- `ALUSrc` is needed before **Mux, ALU, Data Mem, Mux, RegFile**, $880 - 200 - 30 - 120 - 300 - 30 - 100 = 100[ps]$ at most.

  Notice that the most critical signal to generate quickly must be `ALUOp` or `ALUSrc` as they must need shorter time than other signals, and which one is more criticla is determined by **Mux** and **ALU Ctrl**. In that case, because $50(ALUCtrl) > 30(Mux)$, so `ALUOp` is the most critical signal and $80[ps]$ at

most to avoid it being on critical path.

Q9

1. 360[ps]. Determined by the stage that takes the longest time **MEM** in that case.
2. $5 \times 360 = 1800[ps]$.
3. need $4 + 2000 = 2004$ clock cycles, so total time $2004 \times 360 = 721440[ps]$ average $CPI = \frac{2004}{2000} = 1.002$.

VE370 Homework4

Q1

`lw $t0, -4($sp)`

> As the MUX used to choose between data memory and ALU result do not specify ports, here assume 1 as choose from data memory.

1. IF: in PC: the addresss of instruction `lw $t0, -4($sp)`
2. ID: in IF/ID: machine code of instruction `lw $t0, -4($sp)`, 010111 $sp 01000 1111 1111 1111 1100 and $PC+4$.
3. EX: in ID/EX:

WB	***M***	***EX***	***Reg.File out***	
RegWrite: 1	MemRead: 1	RegDst: X, set as 0	Read Data1: R[$sp]	IF/ID.RegisterRs: $sp
MemtoReg:0	MemWrite: 0	ALUSrc: 1	Read Data2: R[$t0]	IF/ID.RegisterRt: $t0
	Branch: 0	ALUOp: 00		IF/ID.RegisterRd: 111 11
				$PC+4$
				$offset(-4)$

4. MEM: in EX/MEM:

WB	***M***	***ALU out***	
RegWrite: 1	MemRead: 1	Zero: not know	Read Data2: R[$t0], as Write Data
MemtoReg:0	MemWrite: 0	Result: R[$sp]-4	$t0, write Register Dst
	Branch: 0		$PC+4+offset(-4)$

5. WB: in MEM/WB:

WB	***Data Mem out***	
RegWrite: 1	Mem[R[$sp]-4]	ALU Result: R[$sp]-4
MemtoReg:0		$t0, write Register Dst

Q2

```
L1: sw $18,—12($8)
L2: lw $3,8($18)
L3: add $6,$3,$3
L4: or $8,$9,$6
```

1. register $3 between instruction L2 and L3

   register $6 between instruction L3 and L4

2. hazard same as in 1. add 2 NOP between instruction L2 and L3, then add 2 NOP between instruction L3 and L4. 12 clock cycles.

3. hazard: register $3 between instruction L2 and L3

   add two NOP between L2 and L3. 10 clock cycles.

4. hazard: register $3 between instruction L2 and L3

   add one NOP between L2 and L3. 9 clock cycles.

Q3

```
    lw  $t0, Offs(Rs)
    bne Rt, $t0, skip
  jr  Rd
skip: ...
```

2. need **add a read register in Reg File**, because read three register (`Rd`, `Rt`, `Rs`). For `Rt` and `Rs` they could be read through the previous read port as the operation just like `lw temp, offset(Rs)`. Then after the Mem stage, `temp=Mem[Offs+Rs]`, **need to compare the value of temp and Rt**. Then, to change the value of PC **need to add R[Rd] as the input of PC MUX and a select signal that has the result of comparison**.

3. a) Control signal `BEQM` test whether we have this instruction and the comparison result. Like Branch, it has an and GATE with another input as the comparison result of `Mem[Offs+Rs]` and `Rt`.

   b) Control signal to select whether the PC should be `R[Rd]`

4. Data hazard could be resolved by adding a forwarding path, from new stage to EX.

   Control hazard created by new PC is unknown until `beqm` is finished. It would be longer as the result is unknown until MEM and comparison is done.

Q4

```
L1: sub $6, $2, $1
L2: lw  $3, 8($6)
L3: lw  $2, 0($6)
L4: or  $3, $5, $3
L5: sw  $3, 0($5)
```

1.

	1	2	3	4	5
L1	IF	ID	EX($6=$2-$1)	MEM	WB
L2		IF	ID($6)	EX	MEM($3=MEM[8+$6])
L3			IF	ID($6)	EX
L4				IF	ID
L5					IF

without hazard detection unit, all will be correct, as no instruction need the register that just be loaded through the `load` instruction. (L3 do not need $3, L4 do not need $2)

2.

	PC Write	IF/ID Write	Hazard	ID/EX Mem Read	IF/ID Rs	IF/ID Rt	ID/EX Rt	ID/EX Rs	EX/MEM Dst	EX/MEM RegWrite	ForwardA	Forward B	MEM/WB Dst	MEM/WB RegWrite
CC1	1	1	x	x	x	x	x	x	x	x	x	x	x	x
CC2	1	1	x	x	$2	$1	x	x	x	x	x	x	x	x
CC3	1	1	0	0	$6	$3	$1	$2	x	x	00	00	x	x
CC4	1	1	0	1	$6	$2	$3	$6	$6	1	10	00	x	x
CC5	1	1	0	1	$3	$5	$2	$6	$3	1	00	00	$6	0

3.

- Check destination register of ID/EX and Regwrite with IF/ID.Rt/Rs, need to add stall when dependence (register $6 between L1 and L2, register $3 between L4 and L5)
- Check destination register of EX/MEM and Regwrite with IF/ID.Rt/Rs, need to add stall when dependence (register $6 between L1 and L3, register $3 between L2 and L4)

VE370 Homework5

```
        lw R2,0(R1)
Label1: beq R2,R0,Label2 ; Not taken once, then taken
        lw R3,0(R2)
        beq R3,R0,Label1 ; Taken
        add R1,R3,R1
Label2: sw R1,0(R2)
```

Q1

1. The control hazard detection unit need to identify whether the current `branch` instruction depends on the previous result. As it is in the ID Stage, the previous one instruction just enters EX (affect R-type and `lw`) and second-previous just enters MEM (affect `lw`).
   - As the first branch instruction `beq R2,R0,Label2` depends on `R2`, which is the previous one instruction, the result of `Mem[0+R1]` will not be produced until two clock cycles later (when `beq` enters). So need to add two NOPS.
   - If the `lw` is the second-previous instruction of `beq`, only one NOP needed.
   - If depends on the previous R-type instruction, need to add one NOP.
2. Assume not taken.

   MEM Stage, need to add two nops for lw data hazard and two times 3 clock cycles flush. Then total

   $6+4+2+6=18$.

   ID Stage, need to add four nops and two times 1 clock cycle flush. Then total $6+4+4+2=16$.

   So we have 2 clock cycles speedup.
3. Register `R2` between `lw R2,0(R1)` and `beq R2,R0,Label2`.

   Register `R3` between `lw R3,0(R2)` and `beq R3,R0,Label1`.
4. Branch in **ID**, then if there is previous dependent instruction and will give the expected result, need to add stall, then the forwarding path
   - *depends on previous R-type:* in MEM / WB stage, need to be forward.

- *depends on previous load*: in WB stage, need to be forward.

Notice that the forwarding unit actually take **the same input** (*MEM/WB.RegWrite, MEM/WB.RegisterRd, EX/MEM.RegWrite, EX/MEM.RegisterRd, ID/EX.RegisterRs, ID/EX.RegisterRt*). Then the **output** should be same as two 2 bit control signal that control two MUXes, as the input of the comparator.

two MUX: selects among value from *register file*, *ALUResult from EX/MEM*, *data from MEM/WB*.

***beq* R2,R0,Label2** will add two stall then the forwarding unit detect the *ID/EX.RegisterRs(`R2`) == MEM/WB.RegisterRd(`R2`), MEM/WB.RegisterRd=1*, no forward from `EX/MEM`. Then the forward A will choose from **data from MEM/WB** (same as before, will be 01). Forward B still 00.

Q2

EX stage: two extra clock cycles.

1. extra CPI is $30\% \times (1 - 45\%) \times 2 = 0.33$.
2. Assume we will not jump. the jump outcome is determined in the `ID` stage, so one extra clock cycle needed. $30\% \times (1 - 55\%) \times 2 + 5\% \times 1 = 0.32$

Q3

Strong not taken: SNT

	T	NT	NT	T	T	T	T	NT	(2)T	NT	NT	T	T	T	T	NT	(3)T
State	SNT	SNT	SNT	NT	T	ST	ST	T	ST	T	NT	T	ST	ST	ST	T	ST
Predict	True	True	False	False	True	True	False	True	False	False	False	True	True	True	False	True	...

Find that after the second eight instructions, it begins to loop. So with repeated loop forever, it will be

$lim_{n\to\infty} \frac{3+4n}{8n} = 0.5$

VE370 Homework6

Yu Xinmiao 518021910792

Q1

Determined in MEM stage.

Select the MUX before PC: 0x80000180.

`IF.Flush` / `ID.Flush` / `EX.Flush` / `Mem.Flush` = 1

EPC will store *address of the instruction* `sw R2, -120(R4)` + *4*

CAUSE will store the cause

Q2

a. temporal locality: `B[I][0]`

b. spatial locality: `A[J][I]`

Q3

1. cache with 8 one-word blocks (index: 3bits, no word offset)

   Byte offset all as 00, omit zeros in **Tag**.

Word Addr	Binary Addr	Tag	Index	Hit/Miss
4	27'b0 100 00	0	100	Miss
12	26'b0 1 100 00	1	100	Miss
33	24'b0 100 001 00	100	001	Miss
6	27'b0 110 00	0	110	Miss
187	22'b0 10111 011 00	10111	011	Miss
65	23'b0 1000 001 00	1000	001	Miss
186	22'b0 10111 010 00	10111	010	Miss
19	25'b0 10 011 00	10	011	Miss
125	23'b0 1111 101 00	1111	101	Miss
43	24'b0 101 011 00	101	011	Miss
152	22'b0 10011 000 00	10011	000	Miss
253	22'b0 11111 101 00	11111	101	Miss

Cache

Index	V	Tag	Data
000	Y	10011	(152)
001	Y	$100 \rightarrow 1000$	$(33) \rightarrow (65)$
010	Y	10111	(186)
011	Y	$10111 \rightarrow 10 \rightarrow 101$	$(187) \rightarrow (19) \rightarrow (43)$
100	Y	$0 \rightarrow 1$	$(4) \rightarrow (12)$
101	Y	$1111 \rightarrow 11111$	$(125) \rightarrow (253)$
110	Y	0	(6)
111	N		

2. cache with 4 two-word blocks (index: 2bits, word offset: 1bit)

   Byte offset all as 00, omit zeros in **Tag**.

Word Addr	Binary Addr	Tag	Index	Word Offset	Hit/Miss
4	27’b0 10 0 00	0	10	0	Miss
12	26’b0 1 10 0 00	1	10	0	Miss
33	24’b0 100 00 1 00	100	00	1	Miss
6	27’b0 11 0 00	0	11	0	Miss
187	22’b0 10111 01 1 00	10111	01	1	Miss
65	23’b0 1000 00 1 00	1000	00	1	Miss
186	22’b0 10111 010 00	10111	01	0	Hit
19	25’b0 10 011 00	10	01	1	Miss
125	23’b0 1111 101 00	1111	10	1	Miss
43	24’b0 101 011 00	101	01	1	Miss
152	22’b0 10011 000 00	10011	00	0	Miss
253	22’b0 11111 101 00	11111	10	1	Miss

Cache

Index	V	Tag	Word0	Word1
00	Y	$100 \rightarrow 1000 \rightarrow 10011$	$(32) \rightarrow (64) \rightarrow (152)$	$(33) \rightarrow (65) \rightarrow (153)$
01	Y	$10111 \rightarrow 10 \rightarrow 101$	$(186) \rightarrow (18) \rightarrow (42)$	$(187) \rightarrow (19) \rightarrow (43)$
10	Y	$0 \rightarrow 1 \rightarrow 1111 \rightarrow 11111$	$(4) \rightarrow (12) \rightarrow (124) \rightarrow (252)$	$(5) \rightarrow (13) \rightarrow (125) \rightarrow (253)$
11	Y	0	(6)	(7)

3. *In terms of miss rate:* **C2 is the best one**
    - C1: 1-word 8 blocks: all miss, miss rate = 1
    - C2: 2-word 4 blocks: one hit (**186**, **187**), miss rate = 1- 1/12 = 0.92
    - C3: 4-word 2 blocks: no hit, because although in the same block, before

next data arrived, (**4**,5,6,**7**) (184,185,**186**,**187**) will not create hit. miss rate = 1

In terms of time: **C2 is the best one**

- C1: stall time = 35 * 12 + 2 * 12 = 444
- C2: stall time = 35 * 11 + 3 * 12 = 421
- C3: stall time = 35 * 12 + 5 * 12 = 480

Q4

1. bits 11-7 (cache index): 32 blocks in one cache

   bits 6-2 (word offset): 32 words in one block

   Tag: 32-12=20 bits

   size of cache: $(32 \times 32 + 20 + 1 + 1) \times 32 = 33472$ bits

2. data: $32 \times 32 \times 32 = 32768$ bits

   Ratio = $\frac{33472}{32768} = 1.021$

3. **5 blocks** are replaced, if replace empty could be considered as replacement. Otherwise as 0 blocks.

Byte addr	Tag	Index	Word Offset	Byte Offset	Hit/Miss
0	0	00000	00000	00	Miss
4(1 00)	0	00000	00001	00	Hit
20(101 00)	0	00000	00101	00	Hit
136(1 00010 00)	0	00001	00010	00	Miss
232(1 11010 00)	0	00001	11010	00	Hit
164(1 01001 00)	0	00001	01001	00	Hit
1024(1000 00000 00)	0	01000	00000	00	Miss
30(111 10)	0	00000	00111	10	Hit
140(1 00011 00)	0	00001	00011	00	Hit
3100(11000 00111 00)	0	11000	00111	00	Miss
176(1 01100 00)	0	00001	01100	00	Hit
2180(10001 00001 00)	0	10001	00001	00	Miss

4. hit ratio = 7 / 12 = 0.583
5. <index, tag, data> **data represent using data address**

   <00000, 0, (0,1,2,3,4,5,6,7,8,....31)>

   <00001, 0, (32,33,....63)>

   ...

   <01000, 0, (256,257,....287)>

   ...

   <10001, 0, (544, ..., 575)>

   ...

   <11000, 0,(768, ..., 799) >

VE370 Homework7

> All data represented using (word address)

Q1

1. 66944 bits

   5 bits index: 32 sets. 2 blocks in each set. *5 bits word offset*: 32 words in each block.

   Each block: 1'b Valid + 1'b Dirty + 20'b Tag + 32 * 32'b Data = 1046bits. *cache:* $32 \times 2 \times 1046 = 66944$ bits.

2. $7/12 = 0.583$

0(m)	4(h)	20(h)	136(m)	232(h)	164(h)	1024(m)	30(h)	140(h)	3100(m)	176(h)	2180(m)
0	100	10100	1 00010 00	1 11010 00	1 01001 00	1000 00000 00	111 10	1 00011 00	11000 00111 00	1 01100 00	10001 00001 00

3. Cache <index, tag, data>

Set Index	Tag	Data
00000	20'b0	(0,1,…, 31)
	Empty	
00001	20'b0	(32, 33, …, 63)
	Empty	
….		
01000	20'b0	(256, 257, …, 287)
	Empty	
...		
10001	20'b0	(544, 545, …, 575)
	Empty	
...		
11000	20'b0	(768, 769, …, 799)
	Empty	
...		

Q2

1. $P_1 : 1/1.18ns = 847MHz.$
   $P_2 : 1/2.22ns = 450MHz$
2. AMAT = Hit time + Miss Rate * Miss Penalty

$$
\begin{aligned}
P_1: &\quad 1.18 + 4.3\% \times 70ns = 4.19 \text{ ns} \\
P_2: &\quad 2.22 + 2.7\% \times 70ns = 4.11 \text{ ns}
\end{aligned}
$$

3.

$$P_1: \quad CPI = 1 + 36\% \times 4.3\% \times (70ns/1.18ns) = 1.92 \text{ clock cycles}$$
$$P_1: \quad CPI = 1 + 36\% \times 2.7\% \times (70ns/2.22ns) = 1.31 \text{ clock cycles}$$

Q3

1. 3 block in one set, 4 sets in cache

byte address 'binary' (data)	**2'b Index**	**27'b Tag**	**3'b Offset**	**hit/miss**
3 '0b11' (0)	00	27'b0	0 11	m
180 '0b10110100' (45)	10	24'b0+101	1 00	m
43 '0b101011' (10)	01	26'b0+1	0 11	m
3 '0b11' (0)	00	27'b0	0 11	h
191 '0b10111111' (47)	11	24'b0+101	1 11	m
89 '0b1011001' (22)	11	25'b0+10	0 01	m
190 '0b10111110' (47)	11	24'b0+101	1 10	h
14 '0b1110' (3)	01	27'b0	1 10	m
181 '0b10110101' (45)	10	24'b0+101	1 01	h
44 '0b101100' (11)	01	26'b0+1	1 00	h
186 '0b10111010' (46)	11	24'b0+101	0 10	h
252 '0b11111100' (63)	11	24'b0+111	1 00	m

Cache, omit 0 in tag

Index	V	D	R	Tag	Data
00	Y	0	1	0	(0, 1)
	N				
	N				
01	Y	0	0	1	(10, 11)
	Y	0	0	0	(2, 3)
	N				
10	Y	0	1	101	(44, 45)
	N				
	N				
11	Y	0	1	101	(46, 47)
	Y	0	0	10	(22, 23)
	Y	0	0	111	(62, 53)

2. 8 blocks in one set, one set in cache. Index / Word offset: 0 bit

byte address 'binary' (data)	0'b Index	30'b Tag	2'b Offset	hit/miss
3 '0b11' (0)		30'b0	11	m
180 '0b10110100' (45)		24'b0+101101	00	m
43 '0b101011' (10)		26'b0+1010	11	m
3 '0b11' (0)		30'b0	11	h
191 '0b10111111' (47)		24'b0+101111	11	m
89 '0b1011001' (22)		25'b0+10110	01	m
190 '0b10111110' (47)		24'b0+101111	10	h
14 '0b1110' (3)		28'b0+11	10	m
181 '0b10110101' (45)		24'b0+101101	01	h
44 '0b101100' (11)		26'b0+1011	00	m
186 '0b10111010' (46)		24'b0+101110	10	m
252 '0b11111100' (63)		24'b0+111111	00	m

Cache, omit zero in tag

V	D	R	Tag	Data
Y	0	1	0	(0)
Y	0	1	101101	(45)
Y	1	0	111111	(63)
Y	0	1	101111	(47)
Y	0	0	10110	(22)
Y	0	0	11	(3)
Y	0	0	1011	(11)
Y	0	0	101110	(46)

3. 2 words in one block, 4 blocks in one set, 1 set in cache. Index: 0 bit, Word

Offset: 1 bit

- **Using LRU**, when no empty slot, replace which one **R==0**.

  Miss Rate = $9/12 = 0.75$

byte address 'binary' (data)	0'b Index	29'b Tag	3'b Offset	hit/miss
3 '0b11' (0)		29'b0	0 11	m
180 '0b10110100' (45)		24'b0+10110	1 00	m
43 '0b101011' (10)		26'b0+101	0 11	m
3 '0b11' (0)		29'b0	0 11	h
191 '0b10111111' (47)		24'b0+10111	1 11	m
89 '0b1011001' (22)		25'b0+1011	0 01	m
190 '0b10111110' (47)		24'b0+10111	1 10	h
14 '0b1110' (3)		28'b0+1	1 10	m
181 '0b10110101' (45)		24'b0+10110	1 01	m
44 '0b101100' (11)		26'b0+101	1 00	m
186 '0b10111010' (46)		24'b0+10111	0 10	h
252 '0b11111100' (63)		24'b0+11111	1 00	m

Cache, omit zero in tag

V	D	R	Tag	Data
Y	0	1	$0 \rightarrow 10110$	$(0,1) \rightarrow (44,45)$
Y	0	0	$10110 \rightarrow 1011 \rightarrow 101$	$(44,45) \rightarrow (22,23) \rightarrow (10,11)$
Y	0	1	$101 \rightarrow 1 \rightarrow 11111$	$(10,11) \rightarrow (2,3) \rightarrow (62,63)$
Y	0	1	10111	(46, 47)

3.

- **Using MRU**, when no empty slot, replace which one **R==1**.

Miss Rate = $9/12 = 0.75$

byte address 'binary' (data)	0'b Index	29'b Tag	3'b Offset	hit/miss
3 '0b11' (0)		29'b0	0 11	m
180 '0b10110100' (45)		24'b0+10110	1 00	m
43 '0b101011' (10)		26'b0+101	0 11	m
3 '0b11' (0)		29'b0	0 11	h
191 '0b10111111' (47)		24'b0+10111	1 11	m
89 '0b1011001' (22)		25'b0+1011	0 01	m
190 '0b10111110' (47)		24'b0+10111	1 10	m
14 '0b1110' (3)		28'b0+1	1 10	m
181 '0b10110101' (45)		24'b0+10110	1 01	h
44 '0b101100' (11)		26'b0+101	1 00	h
186 '0b10111010' (46)		24'b0+10111	0 10	m
252 '0b11111100' (63)		24'b0+11111	1 00	m

Cache, omit zero in tag

V	D	R	Tag	Data
Y	0	0	0	$(0, 1)$
Y	0	0	10110	$(44, 45)$
Y	0	1	$101 \rightarrow 10111 \rightarrow 11111$	$(10, 11) \rightarrow (46, 47) \rightarrow (62, 63)$
Y	0	0	$10111 \rightarrow 1011 \rightarrow 10111 \rightarrow 1$	$(46, 47) \rightarrow (22, 23) \rightarrow (46, 47) \rightarrow (2, 3)$

- Best possible miss rate: $7/12 = 0.583$

VE370 Homework8

Q1

1. $1 \times 1.18 + 7.3\% \times 5.34 + 7.3\% \times 1.5\% \times 70 = 1.65ns$
2. cycle time for memory acess: $1.65ns/1.18ns$

   $CPI = 1 \times 0.64 + (1.65/1.18) \times 0.36 = 1.14$

Q2

> Set have two block, represent with (1) as one block.
>
> In one set, the first one is least used

1. **LRU: 3 hits**

Block Address of Memory	Hit/Miss	Evicted Block	Set 0	Set 1
1	M	Null		(1)
3	M	Null		(1)(3)
5	M	(1)		(3)(5)
1	M	(3)		(5)(1)
3	M	(5)		(1)(3)
1	H	Null		(3)(1)
3	H	Null		(1)(3)
5	M	(1)		(3)(5)
3	H	Null		(5)(3)

2. **MRU: 3hits**

Block Address of Memory	Hit/Miss	Evicted Block	Set 0	Set 1
1	M	Null		(1)
3	M	Null		(1)(3)
5	M	(3)		(1)(5)
1	H	Null		(5)(1)
3	M	(1)		(5)(3)
1	M	(3)		(5)(1)
3	M	(1)		(5)(3)
5	H	Null		(3)(5)
3	H	Null		(5)(3)

3. **Random: 3hits**

Block Address of Memory	Hit/Miss	Evicted Block	Set 0	Set 1
1	M			(1)
3	M			(1)(3)
5	M	(1)		(5)(3)
1	M	(5)		(1)(3)
3	H			(1)(3)
1	H			(1)(3)
3	H			(1)(3)
5	M	(3)		(1)(5)
3	M	(5)		(1)(3)

Q3

> **TLB hit** means Page hit and no Page Fault.

> **Page hit** means TLB miss and no Page Fault.
>
> **Page Fault** means TLB miss and Page miss.

1. virtual address = 4bits virtual page number + 12bits offset

12948(0x3294)	49419(0xc10b)	46814(0xb6de)	13975(0x3697)	40004(0x9c44)	12707(0x31a3)	52236(0xcc0c)
TLB hit	Page Fault	TLB hit	TLB hit	Page Fault	TLB hit	TLB hit

TLB

Valid	Tag	Physical Page Number
1	11	12
1	9	14
1	3	6
1	12	13

Page Table

Valid	Physical Page Number
1	5
0	Disk
0	Disk
1	6
1	9
1	11
0	Disk
1	4
0	Disk
1	14
1	3
1	12
1	13

2. virtual address = 2bits virtual page number + 14bits offset

12948(2'b00)	49419(2'b11)	46814(2'b10)	13975(2'b00)	40004(2'b10)	12707(2'b00)	52236(2'b11)
Page Table hit	TLB hit	Page Fault	TLB hit	TLB hit	TLB hit	TLB hit

TLB

Valid	Tag	Physical Page Number
1	2	13
1	7	4
1	3	6
1	0	5

Page Table

Valid	Physical Page Number
1	5
0	Disk
1	13
1	6
1	9
1	11
0	Disk
1	4
0	Disk
0	Disk
1	3
1	12
0	Disk

3. **Advantages:** more address could be stored in one page, small page table, less page entry, less page fault

   Disadvantages: lots of wasted memory

4. virtual address =4bits virtual page (3bits tag + 1bit set offset) + 12bits offset

   Assume the initial state of TLB same as (1)

12948(0b0011)	49419(0b1100)	46814(0b1011)	13975(0b0011)	40004(0x1001)	12707(0b0011)	52236(0b1100)
Page hit	Page Fault	Page hit	TLB hit	Page Fault	TLB hit	TLB hit

TLB

	Valid	Tag	Physical Page Number
Set0	1	3’b110	13
Set0	1	3’b101	12
Set1	1	3’b100	14
Set1	1	3’b001	6

Page Table

Valid	Physical Page Number
1	5
0	Disk
0	Disk
1	6
1	9
1	11
0	Disk
1	4
0	Disk
1	14
1	3
1	12
1	13