ECE 6913, Fall 2020

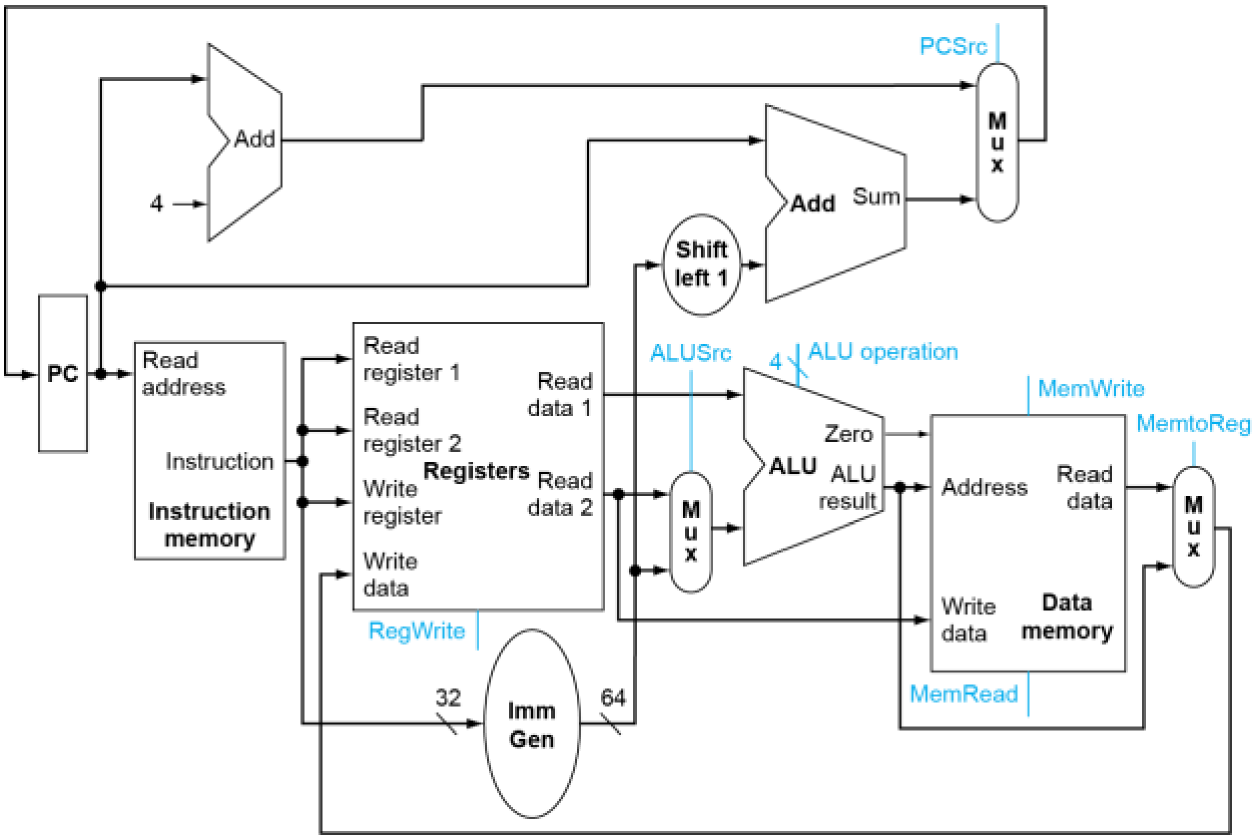
Misc Problems on topics covered on Quiz 2

1.1 Consider a datapath similar to the one in figure below, but for a processor that only has one type of instruction: *unconditional PC-relative branch*. What would the cycle time be for this datapath? How about the case for a *conditional PC-relative branch - What would its cycle time be?*

I-Mem, D-Mem	Register File	Mux	ALU	Adder	Shift left 2	Register Read	Register Setup	Sign Extend	Control
250ps	150ps	25ps	200ps	150ps	20ps	30ps	20ps	50ps	50ps

"Register read" is the time needed after the rising clock edge for the new register value to appear on the output. This value applies to the PC only.

"Register setup time" is the amount of time a register's data input must be stable before the rising edge of the clock. This value applies to ***both the PC and Register File****.*



(i) A *conditional* PC relative branch needs the ALU to perform a subtraction to determine if the source registers identified in the instruction are equal

PC → Instr Memory → Register File → Mux → ALU → Single Gate → Mux → PC Register Setup Time → PC
30ps + 250ps +150ps + 25ps + 200ps + 5ps + 25ps + 20ps = **705ps**

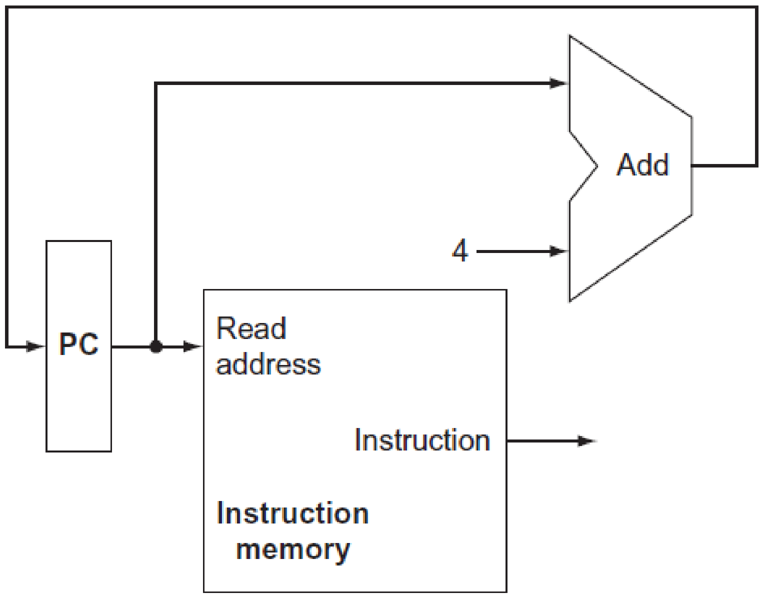
Note that the path to calculate the branch address is faster than the concurrent path above between Register File and PC Mux

(ii) An *unconditional* PC relative branch instruction does not need the ALU to determine if a branch is taken.
The path of execution is:

PC → Instr Memory → Sign-extend → Shift left 2 → Add → Mux → Setup delay for PC Registers →PC

From the above Table, this equals
30ps + 250ps + 50ps + 20ps + 150ps + 25ps + 20ps = **545ps**

1.2 If the only thing we need to do in a processor is fetch consecutive instructions using the figure below, what would the cycle time be?



Fetch of instructions would be dominated by the path from Program Counter to Instruction availability at the output of Instruction Memory. The Read delay of the Program Counter (30ps) following a clock edge added to the access delay of the Instruction memory (250ps) exceeds the concurrent execution of incrementing the current address by 4 and updating the Program Counter with it
30ps + 250ps = **280ps cycle time**

The other path to the adder followed by write into the Program Counter equals
30ps + 150ps + 20ps = 200ps
and is faster, so **the cycle time is limited by the slower 280 ps**

2. Your favorite Computer is described with the following features:
• 95% of all memory accesses are found in the cache.
• Each cache block is two words, and the whole block is read on any miss.
• The processor sends references to its cache at the rate of 10^9 words per second.
• 25% of those references are writes.
• Assume that the memory system can support 10^9 words per second, reads or writes.
• The bus reads or writes a single word at a time (the memory system cannot read or write two words at once).
• Assume at any one time, 30% of the blocks in the cache have been modified.
• The cache uses write allocate on a write miss.

You are considering adding a peripheral to the system, and *you want to know how much of the memory system bandwidth is already used*.

(a) Calculate the percentage of memory system bandwidth used assuming the cache is Write Back.
(b) Calculate the percentage of memory system bandwidth used assuming the cache is Write Through.
Be sure to state your assumptions.

We know:

* Miss rate = 0.05

* Block size = 2 words (8 bytes)

* Frequency of memory operations from processor = 109

* Frequency of writes from processor = 0.25 ∗ 109

* Bus can only transfer one word at a time to/from processor/memory

* On average 30% of blocks in the cache have been modified (must be written back in the case of the write back cache)

* Cache is write allocate

So:

Fraction of read hits = 0.75 ∗ 0.95 = 0.7125

Fraction of read misses = 0.75 ∗ 0.05 = 0.0375

Fraction of write hits = 0.25 ∗ 0.95 = 0.2375

Fraction of write misses = 0.25 ∗ 0.05 = 0.0125

2 (b) Write through cache

- On a read hit there is no memory access
- On a read miss memory must send two words to the cache
- On a write hit the cache must send a word to memory
- On a write miss memory must send two words to the cache, and then the cache must send a word to memory

Thus:

Average words transferred = $0.7125 * 0 + 0.0375 * 2 + 0.2375 * 1 + 0.0125 * 3 = 0.35$

Average bandwidth used = $0.35 * 10^9$

Fraction of bandwidth used =

$[0.35 \times 10^9] / 10^9$

$= 0.35$ (1)

2(a) Write back cache

On a read hit there is no memory access

On a read miss:

1. If replaced line is modified then cache must send two words to memory, and then

memory must send two words to the cache

2. If replaced line is clean then memory must send two words to the cache

On a write hit there is no memory access

On a write miss:

1. If replaced line is modified then cache must send two words to memory, and then memory must send two words to the cache

2. If replaced line is clean then memory must send two words to the cache

Thus:

Average words transferred = $0.7125 * 0 + 0.0375 * (0.7 * 2 + 0.3 * 4) + 0.2375 * 0 + 0.0125 *$

$(0.7 * 2 + 0.3 * 4) = 0.13$

Average bandwidth used = $0.13 * 10^9$

Fraction of bandwidth used =

$0.13 \times 10^9/10^9$

$= 0.13$ (2)

Comparing 1 and 2 we notice that the write through cache uses more than twice the cache-memory bandwidth of the write back cache.

3. Assume we have a computer where the CPI is 1.0 when all memory accesses (including data and instruction accesses) hit in the cache. The cache is a unified (data + instruction) cache of size 256 KB, 4-way set associative, with a block size of 64 bytes. The data accesses (loads and stores) constitute 50% of the instructions. The unified cache has a miss penalty of 25 clock cycles and a miss rate of 2%. Assume 32-bit instruction and data addresses.

3.1 What are the Number of bits used for block offset?
Block size = 64 bytes, so 6 bits decode one of 64 bytes in a Block. Number of bits used for Block offset = 6

3.2 What are the Number of sets in the cache?
256KB /(64B x 4way) = 1K sets

3.3 What are the Number of bits for the cache index?
bits for the cache index pick one of all available sets ➔10 bits

3.4 What are the Number of bits for the tag?
Address has a total of 32 bits of which 6 bits used for Block offset and 10 bits used for the cache index. So, 32 – 16 = 16 bits for the Tag

3.5 Calculate the number of Stall Cycles per instruction
CPI for a computer that always hits = 1
Stall_Cycles_per_instruction = [Memory accesses per instr] x MR x MP
Memory accesses per instr. = 1 (for Instruction) + 0.5 (for Data) = 1.5

SCPI= 1.5 mem access/instruction x 0.02 misses/access x 25 cycles = 0.75 cycles/instruction

3.6 How much faster would the computer be if all memory accesses were cache hits?

1.75/1 = 1.75 faster

4. Consider the following RISC V Instruction sequence executing in a 5-stage pipeline:

```
or   x13, x12, x11
ld   x10, 0(x13)
ld   x11, 8(x13)
add  x12, x10, x11
subi x13, x12, 16
```

4.1 Identify all of the data hazards and their resolution with NOPs assuming no forwarding or hazard detection hardware is being used

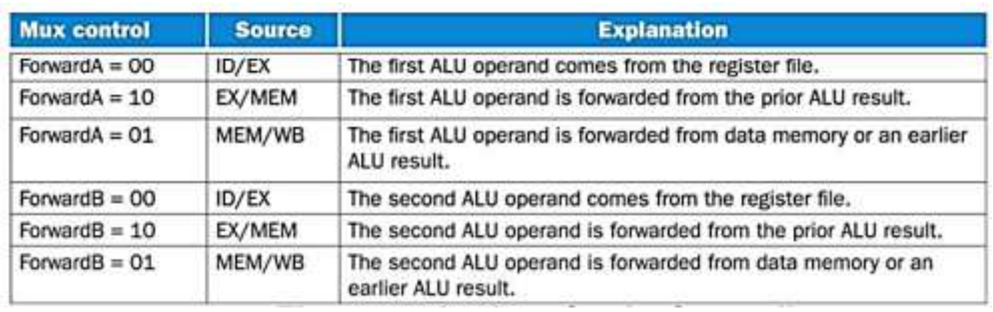
Hazards identified:

```
or   x13, x12, x11
ld   x10, 0(x13)          EX to 1st RAW Hazard
ld   x11, 8(x13)          EX to 2nd RAW Hazard
add  x12, x10, x11        MEM to 1st RAW [load-use-data] & MEM to 2nd Hazards
subi x13, x12, 16         Ex to 1st RAW Hazard
```

NOPS introduced to resolve Hazards:

```
or   x13, x12, x11
NOPS
NOPS
ld   x10, 0(x13)          EX to 1st RAW Hazard resolution with 2 NOPs
ld   x11, 8(x13)          EX to 2nd RAW Hazard resolved as well from above 2 NOPs
NOPS
NOPS
add  x12, x10, x11        MEM to 1st RAW [load-use-data] & MEM to 2nd Hazards
                          resolved with 2 NOPs
NOPS
NOPS
subi x13, x12, 16         Ex to 1st only RAW Hazard resolved with 2 NOPs
```

4.2 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* in Figure below.



Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

	Clock Cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	**or**	IF	ID	**EX**	MEM	WB					
2	**ld**		IF	ID	**EX**	MEM	WB				
3	**ld**			IF	ID	EX	MEM	WB			
4	**NOP**	*mandatory NOP for which no forwarding solution possible: load-data-use*									
5	**add**					IF	ID	EX	MEM	WB	
6	**subi**						IF	ID	EX	MEM	WB

(1)	A=x	B=x	*(no instruction in* EX *stage yet)*
(2)	A=x	B=x	*(no instruction in* EX *stage yet)*
(3)	A=0	B=0	*(both operands of the* or *instruction:* x11,x12 *come from Reg File)*
(4)	A=2	B=0	*(base (*RS1*) in first* ld *(*x13*) taken from EX/MEM of previous instruction)*
(5)	A=1	B=0	*(base (*RS1*) in 2nd* ld *(*x13*) taken from MEM/WB of a previous instruction)*
(6)	A=x	B=x	*(no instruction in* EX *stage yet because NOP introduced to resolve MEM to 1st*
(7)	A=0	B=1	*(*RS2 *in the* add *instruction is* x11 *which is forwarded from MEM/WB of 2nd* ld*, the result of the 1st* ld *(*x10*) has already been written into Reg File in CC 6 - so, no forwarding necessary for first operand)*
(8)	A=1	B=0	*(RS1 of* subi *instruction forwarded from EX/MEM of* add *instruction)*

5. In the conventional fully bypassed 5-stage pipeline discussed in class, we were able to make the assumption that the ALU is able to complete in one cycle because we assumed integer operations. For this problem, we will assume that the ALU takes two cycles to complete. In other words, *the ALU is pipelined itself, making the entire pipeline 6 cycles*. The ALU only *generates a result after 2 cycles* (i.e., there is no way to extract any meaningful result after the ALU's first cycle). Memory (instruction and data) still returns data after only one cycle. You may ignore branch and jump instructions in this problem.

As a first step, we will examine how this changes data hazards. For this question, we will examine only the ALU-ALU read after write (RAW) hazard where the two instructions are consecutive:

```
ADD x1, x2, x3          # x1 <= x2 + x3

ADD x5, x4, x1          # x5 <= x4 + x1
```

(i) Describe *how would you resolve this hazard with the minimum number of bubbles (if any) using a combination of data forwarding and stalls (if necessary).*

(ii) Then fill the following timing diagram to illustrate how the pipeline will behave, and show where data forwarding happens, if at all, by drawing an arrow between the two stages that participate in it.

	CC0	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
ADD x1, x2, x3	IF	ID	ALU 1	ALU 2	MEM	WB			
NOP									
ADD x5, x4, x1			IF	ID	ALU 1	ALU 2	MEM	WB	

Data forwarding alone does not suffice here because by the time the first ALU instruction completes the second ALU (execute) step, the second ALU instruction is already in the first ALU step and thus it's too late.

Therefore, a NOP between the 2 instructions, as well as data forwarding between the second ALU step and the decode step.

6. Indicate if the following modifications (A,B,C) will cause each of the three metrics (three rightmost columns) to *increase*, *decrease*, or have *no effect*. Explain your reasoning

Assume the initial machine is pipelined. Also assume that any modification is done in a way that preserves correctness and maintains efficiency, but that the rest of the machine remains unchanged.

		Instructions/Program	**CPI (Cycles/Instruction)**	**Circuit complexity**
A	Replace the 2 operand ALU with a 3 operand one and add 3 operand register-register instructions to the ISA (for example, `ADD rs1,rs2,rs3,rd`)	**Decrease** The new instructions will replace any two-instruction sequence that accomplished the same, such as `ADD rs1, rs2, rd` `ADD rs3, rd, rd`	**The same** The ALU will perform the three-way addition in one cycle. Also acceptable increase because more RAW	**Increase** Three-operand ALU is more complex than a two-operand one
B	Use the same ALU for instructions and for incrementing the PC by 4	**The same** No difference to instructions	**Increase** All instructions now use the same ALU ALU operations now have to stall	**Decrease** Now there is one less adder/ALU cycle
C	Increase the number of user registers from 32 to 64	**The same or decrease** If the more registers enable the Compiler to avoid loads and stores, Decrease. Otherwise the same.	**The same** Does not affect the actions of each instructions	**Increase** More registers complicate the register file

7. In general, cache access time is proportional to capacity. Assume that main memory accesses take 100 ns and that 40% of all instructions access data memory.

The following table shows data for L1 caches attached to each of two processors, P1 and P2

	L1 Size	L1 Miss Rate	L1 Hit Time
P1	4KiB	10%	0.75ns
P2	8KiB	7%	1.0ns

7.1 Assuming that the L1 hit time determines the cycle times for P1 and P2, what are their respective clock rates?

P1: 1.33GHz

P2: 1GHz

7.2 What is the Average Memory Access Time for P1 and P2 (in cycles)?

P1: AMAT = 0.75ns + 100ns x 10% = 10.75ns

P2: AMAT = 1ns + 100ns x 7% = 8ns

7.3 Assuming a base CPI of 1.5 without any memory stalls, *what is the total CPI for P1 and P2? Which processor is faster?* (When we say a "base CPI of 1.5", we mean that 2 instructions complete in three cycles, unless either the instruction access or the data access causes a cache miss.)

Total CPI for P1= 1.5 + [(100x0.1/0.75ns)] x 0.4 = 6.83

Total CPI or P2 = 1.5 + [(100 x 0.07)/1.0] x 0.4 = 4.3

7.4 Consider the addition of an L2 cache to P1 (to presumably make up for its limited L1 cache capacity). Use the L1 cache capacities and hit times from the previous table when solving these problems. The L2 miss rate indicated is its local miss rate.

L2 Size	L2 Miss Rate	L2 Hit Time
4MiB	90%	10ns

What is the AMAT for P1 with the addition of an L2 cache? Is the AMAT better or worse with the L2 cache?

P1 AMAT: 0.75ns + 0.1 x (10ns + 0.9 x 100ns) = 10.75 ns

AMAT is the same

7.5 Assuming a base CPI of 1.5 without any memory stalls, *what is the total CPI for P1 with the addition of an L2 cache?*

CPI = 1.5 + 0.4 x [0.1(10 + 0.9 x 100)/0.75] = 6.83

8. Mark whether the following modifications to cache parameters will cause each of the categories to increase, decrease, or whether the modification will have no effect. You can assume the baseline cache is set associative. Explain your reasoning. Assume that in each case the other cache parameters (number of sets, number of ways, number of bytes/line) and the rest of the machine design remain the same

	compulsory misses	conflict misses	capacity misses
increasing number of sets	no effect block size is constant	decreases more sets are available for data, so there is less of a chance for two ops to collide and evict one another	decreases capacity increases
increasing number of ways	no effect block size is constant	decreases there are more ways available for data to be placed into	decreases capacity increases
increasing number of bytes per line	decreases more data is brought in on a given cache miss	no effect associativity and number of sets is constant	decreases capacity increases

9. Assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows for the processor shown in the Figure below:

add	addi	not	beq	lw	sw
17%	18%	5%	25%	25%	10%

9.1 In what fraction of all cycles is the shift-left 2 unit used?

The shift left-2 unit is used only for branch instructions: 25%

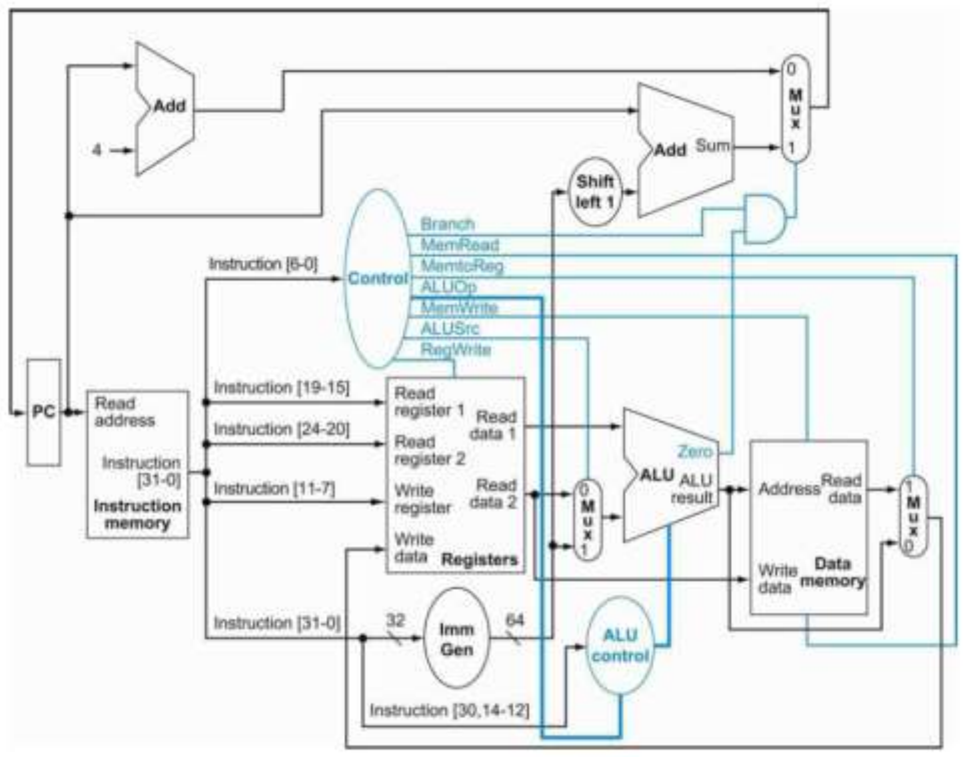
9.2 In what fraction of all cycles is the input of the sign-extend circuit needed?

addi, beq, lw, sw: 18 + 25 + 25 + 10 = 78%

9.3 What is the utilization of the *second* input data port of the ALU?

The second input port of the ALU is used in all instructions except the unconditional branch instruction (jal for example) where the ALU does not need to determine if a branch is to be taken.

add, addi, not, beq, lw, sw = 100%


Add
4
Add
Sum
Shift left 1
M u x
0
1
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [6-0]
PC
Read address
Instruction [31-0]
Instruction memory
Instruction [19-15]
Instruction [24-20]
Instruction [11-7]
Instruction [31-0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
32
64
Imm Gen
Instruction [30,14-12]
ALU control
ALU
Zero
ALU result
Address
Read data
Write data
Data memory