Miscellaneous

- Make Sure understand C
- Can convert binary to decimal and vice versa

How C represents Signed Integers

1. Sign-and-Magnitude

   First bit is sign bit
   Remainder is magnitude

2. 1s complement

Binary	n-bit 1s complement
+X	X
-X	2^n – X - 1

Example:
12 = $(00001100)_2$ = $(00001100)_{1s}$
-12 = 2^n – X – 1 = 243 = $(11110011)_{1s}$
Flip all the bits

Addition:

- Perform binary addition
- move last carry to result
- check overflow

3. 2s complement

Binary	n-bit 2s complement
+X	X
-X	2^n – X

Example:
3-bit
3 = $(011)_2$ = $(011)_{2s}$
-3 = 2^3 – 3 = 5 = $(101)_{2s}$
Starting from the right, flip all the bits after the first 1

Addition:

- Perform binary addition
- Ignore the carry out
- Check overflow

4. Excess-k

   Add K to a value x, to represent in binary
   Excess-8
   -8 => -8 + 8 = 0 = 0000
   0 => 0 + 8 = 8 = 1000

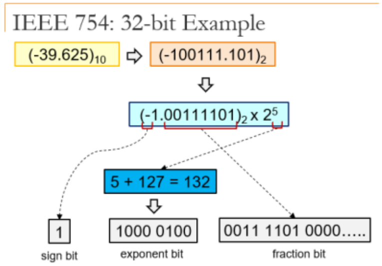
IEEE-754

For 32 bit

Sign bit	1 bit	0 = +ve, 1 = -ve
Exponent	8 bit	Excess 127
Mantissa	23 bit	Nominalized to 1.x, Mantissa is the x

For 64 bit

Sign bit	1 bit	
Exponent	11 bit	
Mantissa	52 bit	



MIPS

NOT: use nor

If condition, jump

```
$s0 < $s1
Slt $t0, $s0, $s1
bne $t0, $zero, dest
```

```
$s0 > $s1
Slt $t0, $s1, $s0
bne $t0, $zero, dest
```

```
$s0 <= $s1
Slt $t0, $s1, $s0
beq $t0, $zero, dest
```

```
$s0 >= $s1
Slt $t0, $s0, $s1
beq $t0, $zero, dest
```

Pipelining

Read after Write
Forwarding

To reduce stalls in loop (beq)

- Early Bird
  Add an adder in Decode stage, use this value to decide, then modify forwarding using pipeline register
- Branch Prediction
  Gambles, and does the instruction, if wrong prediction, flushes out
- Delayed Branch
  Rearranges the instruction, to do while waiting for response, done by compiler

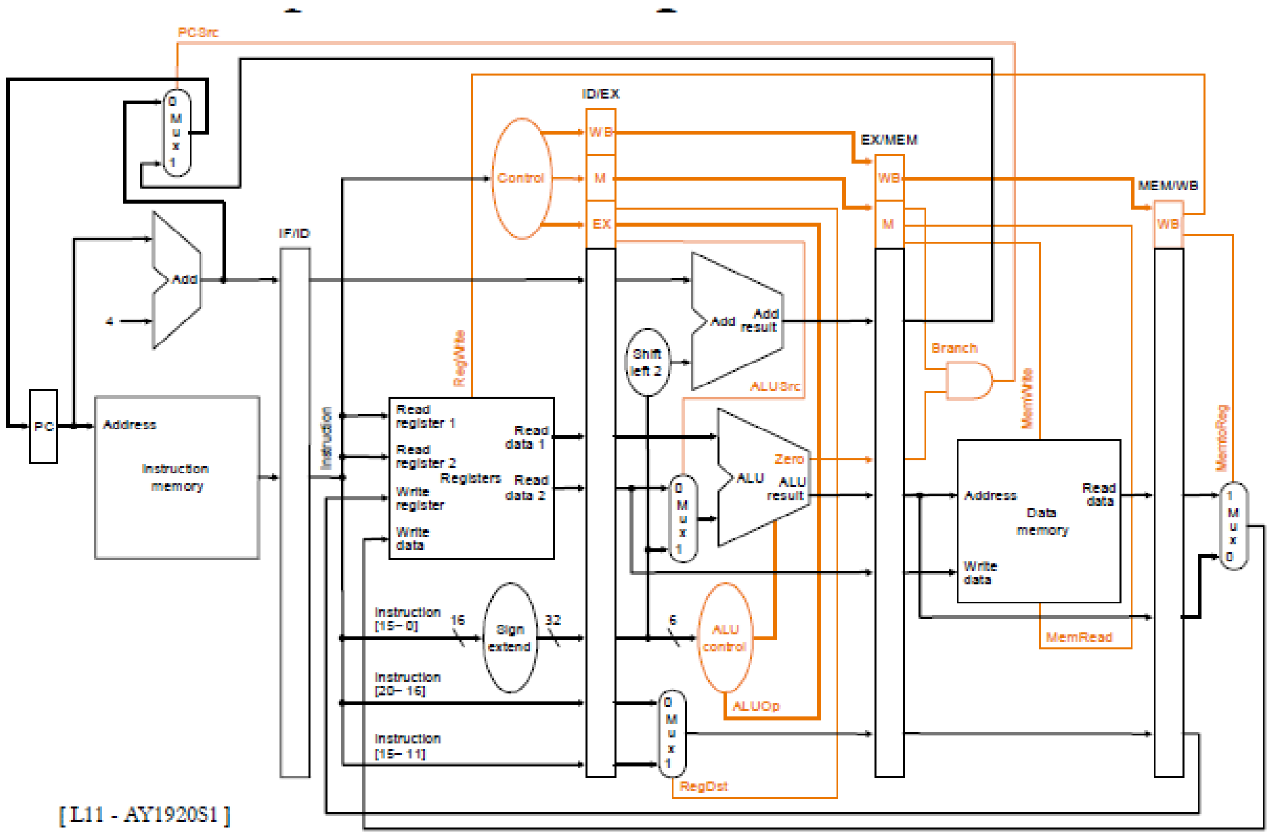
Processor Controls Summary

	RegDst	ALUSrc	MemTo Reg	Reg Write	Mem Read	Mem Write	Branch	ALUop	
								Op1	Op2
R- Type	1	0	0	1	0	0	0	1	0
Lw	0	1	1	1	1	0	0	0	0
Sw	X	1	X	0	0	1	0	0	0
Beq	X	0	X	0	0	0	1	0	1

Opcode	ALUop	Instruction Operation	Funct Field from MIPS	ALU Action	ALU Control
Lw	00	Load Word	XXXXXX	Add	0010
Sw	00	Store Word	XXXXXX	Add	0010
Beq	01	Branch equal	XXXXXX	Sub	0110
R-Type	10	Add	10 0000	Add	0010
R-Type	10	Subtract	10 0010	Sub	0110
R-Type	10	And	10 0100	And	0000
R-Type	10	Or	10 0101	Or	0001
R-Type	10	Slt	10 1010	Slt	0111
R-Type	10	NOR	10 0111	NOR	1100

Processor control uses Opcode to generate ALUop, passed to ALU CONTROL UNIT

ALU CONTROL UNIT uses ALUop and funct field to procude ALU Control



[L11 - AY1920S1]

Last Mux, MemToReg 1 is above, 0 is below

MIPS Assembly Language		
R - Type	Opcode[6] rs[5] rt[5] rd[5] shamt[5] funct[6]	
Addition/subtraction	add / sub rd rs rt	Rd = rs + rt
And / or / xor / nor	And rd rs rt	Rd = rs [op] rt
Shift left / right logical	Sll rd rt shamt	Rd = rt << / >> shamt
Set less than	Slt rd rs rt	Rd ←0/1←rs < rt
I – Type	Opcode[6] rs[5] rt[5] immd[16]	
Add Immediate	Addi rt rs Immd	Rt = rs + immd
Load / store word / byte	lw rt Immd(rs)	Load the word
Branch equal / not equal	Beq rs rt Immd	If rs == rt, go to pc + 4 + immd * 4
J – Type	Opcode[6] address[26]	
jump	J addr	JUMP to address

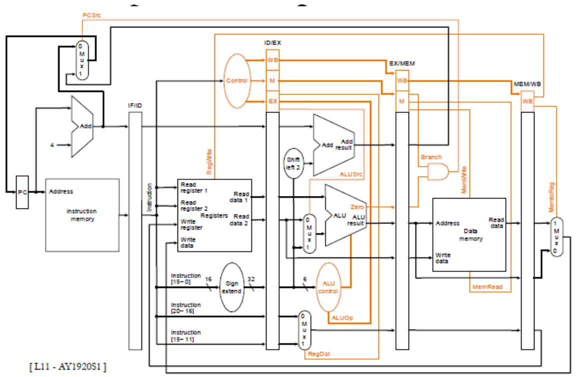
General Purpose Registry

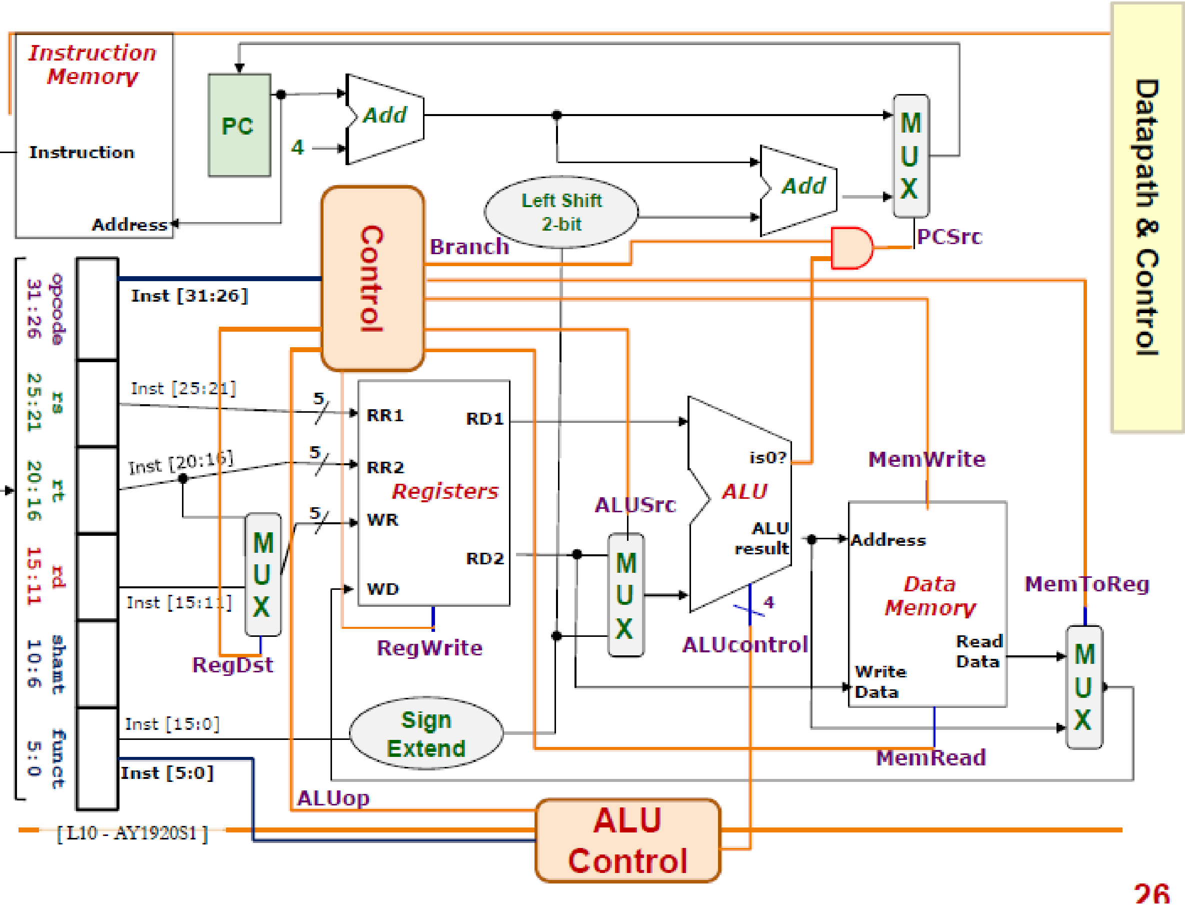
Name	Register number	Usage
$zero	0	Constant value 0
$v0 - $v1	2 – 3	Values for results and expression evaluation
$a0 - $a3	4 – 7	Arguments
$t0 - $t7	8 – 15	Temporaries
$s0 - $s7	16 – 23	Program Variables
$t8 - $t9	24 – 25	More Temporaries
$gp	28	Global pointer
$sp	29	Stack pointer
$fp	30	Flame pointer
$ra	31	Return address

$at (register 1) is reserved for assembler

$k0 - $k1 (register 26 – 27) are reserved for OS

MIPS Assembly Language			
Category	Instruction	Example	Meaning
Arithmetic	Add	`Add $rd, $rs, $Srt`	`$rd = $rs + $rt`
	Subtract	`sub $rd, $rs, $rt`	`$rd = $rs - $rt`
	Add Immediate	`addi $rt, $rs Immd`	`$rt = $rs + Immd`
Data Transfer	Load Word	`lw $s1, 100($s2)`	`$s1 =` Memory[`$s2` + 100]
	Store Word	`sw $s1, 100($s2)`	Memory[`$s2` + 100] = $s1
	Load Byte	`lb $s1, 100($s2)`	`$s1 =` Memory[`$s2` + 100]
	Store Byte	`sb $s1, 100($s2)`	Memory[`$s2` + 100] = $s1
	Load Upper Immediate	`lui $s1, 100`	$s1 = 100 * 2^{16}
Conditional Branch	Branch on not Equal	`beq $s1, $s2, 25`	if (`$s1 == $s2`) go to PC + 4 + 100
	branch on not equal	`bne $s1, $s2, 25`	if (`$s1 != $s2`) go to PC + 4 + 100
	Set on Less than	`slt $s1, $s2, $s3`	if (`$s2 < $s3`) `$s1` = 1; else `$s1` = 0
	Set Less than Immediate	`slti $s1, $s2, 100`	if (`$s2 < 100`) `$s1` = 1; else `$s1` = 0
Unconditional Branch	Jump	`j 2500`	go to 10000
	Jump Register	`jr $ra`	go to `$ra`
	Jump and Link	`jal 2500`	`$ra` = PC + 4; go to 10000


IF/ID
ID/EX
EX/MEM
MEM/WB
[L11 - AY1920S1]


Datapath & Control
Instruction Memory
Instruction
Address
PC
4
Add
Add
M U X
PCSrc
Left Shift 2-bit
Control
Branch
opcode 31:26
rs 25:21
rt 20:16
rd 15:11
shamt 10:6
funct 5:0
Inst [31:26]
Inst [25:21]
Inst [20:16]
Inst [15:11]
Inst [15:0]
Inst [5:0]
5
5
5
RR1
RR2
WR
WD
RD1
RD2
Registers
M U X
RegDst
RegWrite
Sign Extend
ALUSrc
M U X
is0?
ALU
ALU result
4
ALUcontrol
MemWrite
Address
Data Memory
Write Data
Read Data
MemRead
MemToReg
M U X
ALUop
ALU Control
[L10 - AY1920S1]
26