Data Representation & Number Systems

Base-X Conversion

Integers (repeated ÷ by 2)

2	x
2	y R 1
2	z R 0
	⋮
2	1 R 0
	0 R 1

LSB ↑ MSB

Fractions (repeated ×1 by 2)

	Carry
2(0.abc) = 0.xyz	0
2(0.xyz) = 1.def	1
2(0.def) = 1.hjk	1
⋮	
2(0.5) = 1.00	1

MSB ↓ LSB

Data

bit (b): 0, 1
byte (B): 8b
word: 32b / 4B

N bits: 2^N values
M values: $\lceil \log_2 M \rceil$ bits

Binary → Octal / Hexadecimal

(from right ←)

$(x)_8$: 010 | 000 | 011 | 000 = 20140_8
$(x)_{16}$: 010 | 0000 | 0110 | 0000 = 2060_{16}

Integers

	Rep	+ → −	Range	Zeros
	S-M	Flip MSB	$-(2^{n-1}-1)$, $2^{n-1}-1$	+/-
2^n-x-1 ←	1s	Flip All	$-(2^{n-1}-1)$, $2^{n-1}-1$	+/-
2^n-x ←	2s	← till first 1, flip rest	-2^{n-1}, $2^{n-1}-1$	+

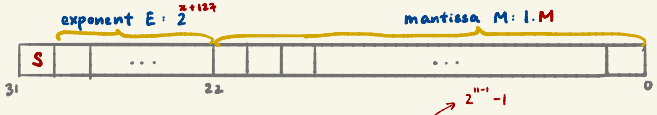
Overflow $A+B$: $(MSB_A == MSB_B)\ !=\ MSB_{A+B}$
1s : Add carry to LSB_{A+B} of result, check for overflow
2s : Ignore carry, check for overflow

IEEE-754

$(S) \times (1.M) \times 2^{E-127}$

$2^{8-1}-1$

exponent E : 2^{x+127} | mantissa M : 1.M



$2^{11-1}-1$

64b : M : 0 to 51, E : 52 to 62, Bias : 1023

Misc

n	0b	0x	2^n
1	1	1	2
2	10	2	4
3	11	3	8
4	100	4	16
5	101	5	32
6	110	6	64
7	111	7	128
8	1000	8	256
9	1001	9	512
10	1010	A	1024
11	1011	B	2048
12	1100	C	4096
13	1101	D	8192
14	1110	E	16384
15	1111	F	32768

C Programming

Primitives

		Range	
Type	Size (b)	Unsigned	Signed
char	8	[0, 255]	[-128, 127]
short	8	[0, 255]	[-128, 127]
int	32	$[0, 2^{31}-1]$	$[-2^{31}, 2^{31}-1]$
long	64	$[0, 2^{64}-1]$	$[-2^{63}, 2^{63}-1]$
float	32	–	–
double	64	–	–

Casting

(cast_type) var

char → int → float → double
no loss →
← loss

Auto-promotion / demotion

```
2 / 5 == 0;
2.0 / 5 == 0.4;
int x = 0.4; // → 0
double y = 5; // → 5.0
```

Pointers

```
int x = 5;
int *y = &x;        // Pointer declaration
print(x);           // 5
print(*y);          // 5   Dereference
print(&x); // addr(x)   Go to address of
print(y); // addr(x)
```

Parameter-Passing

By-value : primitives (int, char, ...), structures
By-address : arrays (strings) → [ ], " "
scanf

```
int x;
scanf("%d", &x); // address
```

Arrays

```
int arr[5] = {1, 2, 3, 4, 5}
```

i	Value	Address
0	1	0x100
1	2	0x104
2	3	0x108
3	4	0x112
4	5	0x116

```
*(arr+2) == arr[2];
arr+2 == &(arr[2]);
arr == &arr[0];
```

Can be passed as parameter, but not as return type

Strings

"Hello" : `char str[6] = "Hello";` (string terminator)
// {'H', 'e', 'l', 'l', 'o', '\0'}

Structures

Defining a new structure:

```
struct struct_name {
    datatype var1;   // one or more fields
    [datatype var2;]
}
```

Ways to instantiate:

```
struct_name str1 = {1};
struct_name str2; str2.var1 = 1;   ==
```

Ways to reference field:

```
struct_name *p = &str1; // pointer of struct_name
(*p).var = 1;
p->var = 1;   ==
```

MIPS

Array and Loop: Version 1.0 — index

Address of A[] → $t0 Result → $t8 i → $t1	Comments
`addi $t8, $zero, 0`	
`addi $t1, $zero, 0`	
`addi $t2, $zero, 40`	`# end point`
`loop: bge $t1, $t2, end`	
`sll $t3, $t1, 2`	`# i x 4`
`add $t4, $t0, $t3`	`# &A[i]`
`lw $t5, 0($t4)`	`# $t3 ← A[i]`
`bne $t5, $zero, skip`	
`addi $t8, $t8, 1`	`# result++`
`skip: addi $t1, $t1, 1`	`# i++`
`j loop`	
`end:`	

Array and Loop: Version 2.0 — pointer

Address of A[] → $t0 Result → $t8 &A[i] → $t1	Comments
`addi $t8, $zero, 0`	
`addi $t1, $t0, 0`	`# addr of current item`
`addi $t2, $t0, 160`	`# &A[40]`
`loop: bge $t1, $t2, end`	`# comparing address!`
`lw $t3, 0($t1)`	`# $t3 ← A[i]`
`bne $t3, $zero, skip`	
`addi $t8, $t8, 1`	`# result++`
`skip: addi $t1, $t1, 4`	`# move to next item`
`j loop`	
`end:`	

Key Pitfalls

Load 32b immediate

```
Lui $s0, immd[32:16]
ori $s0, immd[15:0]
```

Branch-equal immediate

Immd no. of instructions:
AFTER next instruction

Immd size: 16b → $2^{16}-1$
∴ 2^{16} instructions from beq/bne

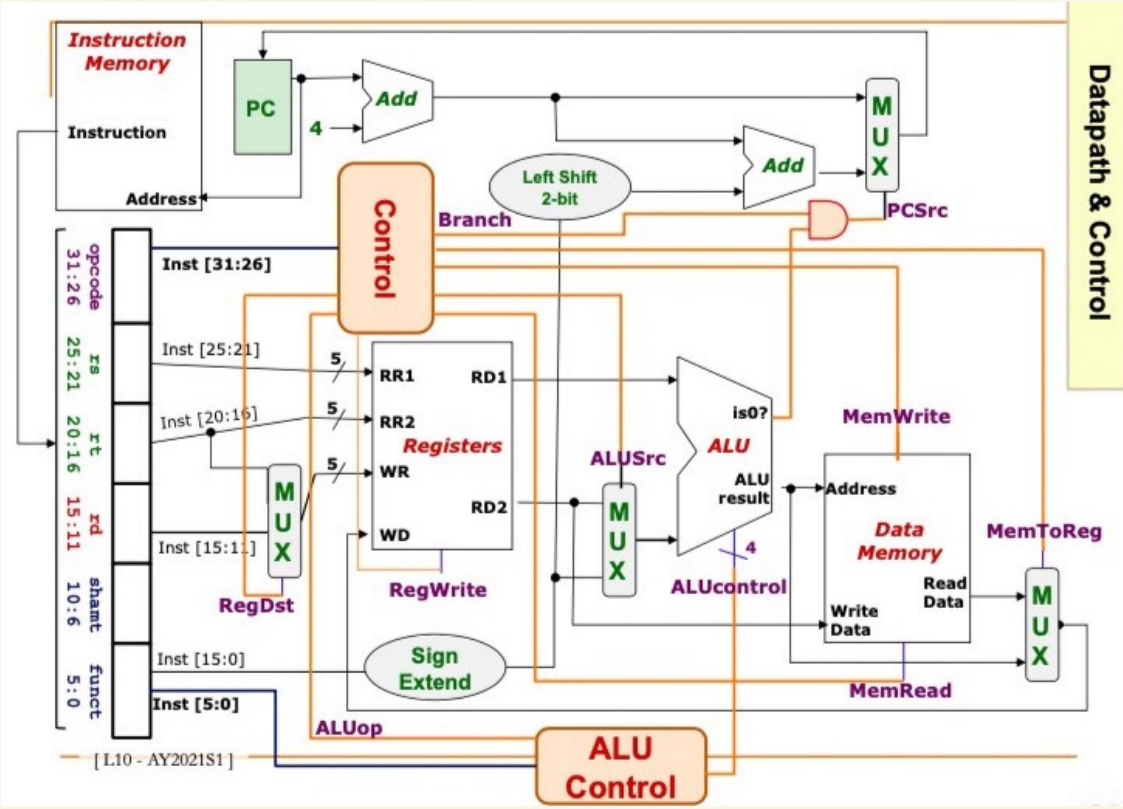
J-format immediate

Immd: 26b 'address'

Word-aligned:

- 2 LSB disregarded
- 4 MSB disregarded

∴ Immd: Addr[27:2]



Encoding

Branching

HLL code	if (x < y) { next; } branch;			if (x ≤ y) { next; } branch;			if (x > y) { next; } branch;			if (x ≥ y) { next; } branch;			if (x == y) { next; } branch;			if (x != y) { next; } branch;		
Possible x and y	x < y	x = y	x > y	x < y	x = y	x > y	x < y	x = y	x > y	x < y	x = y	x > y	x < y	x = y	x > y	x < y	x = y	x > y
Truth values for above	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
slt j x y	1	0	0	1	0	0	1	0	0	1	0	0						
slt k y x	0	0	1	0	0	1	0	0	1	0	0	1						
beq j 0 branch	Next	Branch	Branch	Next	Branch	Branch	Next	Branch	Branch	Next	Branch	Branch						
bne j 0 branch	Branch	Next	Next	Branch	Next	Next	Branch	Next	Next	Branch	Next	Next						
beq k 0 branch	Branch	Branch	Next	Branch	Branch	Next	Branch	Branch	Next	Branch	Branch	Next						
bne k 0 branch	Next	Next	Branch	Next	Next	Branch	Next	Next	Branch	Next	Next	Branch						
beq x y branch													Next	Branch	Next	Next	Branch	Next
bne x y branch													Branch	Next	Branch	Branch	Next	Branch

Control Signals

	EX				MEM			WB	
	RegDst	ALUSrc	ALUOp op1	ALUOp op0	Mem Read	Mem Write	Branch	MemTo Reg	Reg Write
R-type	1	0	1	0	0	0	0	0	1
lw	0	1	0	0	1	0	0	1	1
sw	X	1	0	0	0	1	0	X	0
beq	X	0	0	1	0	0	1	X	0

ALU Control

ALUcontrol	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	slt
1100	NOR