CS2100 Midterms CheatSheet Adapted 23/24

C Language

~ is NOT, ^ is XOR (Bitwise Operator)
! is boolean operator, not bitwise operator
& is bitwise operator, && is boolean operator. Same for |.
%lf is used when you want to scan double
\t is used in printing to print a tab

Int / float: 4 bytes, double: 8 bytes, char: 1 byte.
& -> Address operator. &x gets address of variable x.
* -> declares pointer / dereferencing of pointer.
*x = 32 (following pointer to get value)
Upper / lower case alphabet letter in ASCII differs by **32**.

Syntax of Structure

```
typedef struct {
   members…..
} <structure name>;
```

Use . if you want to access the members of a structure.

Can do assignments with structure, if assign one structure to another structure variable, will copy all members of structure into respective members of structure of structure variable.

(*player_ptr).name /*legal*/
*player_ptr.name /*illegal as treated as *(player_ptr.name)/
player_ptr->name /*legal*/
Declaring an array of pointers: type *a[]

Number System

Conversion from decimal to base-R

Whole number: repeated division by R (copy resultant digits down to up)
Fractional number: repeated multiplication by R (copy resultant digits up to down)

*Negation of a number (**Negated value)***

1s: $-x = 2^n - x - 1$ (n is number of bits of x in binary)
To get negation of x: invert all the bits of x (same for fraction)

2s: $-x = 2^n - x$ (n is number of bits of x in binary)
To get negation of x: invert all the bits of x and **add 1**
(if fraction, same thing, but plus 1 at the **rightmost digit** after the ".")

Addition

2s: Perform addition as usual, **ignore carry out of the MSB**, check for overflow (occur when the resultant sign bit is different from that of the operands in the case where both operands have the same sign)
1s: Same as above, **however if there is a carry out of the MSB, add 1 to the rightmost digit (**same in fractional num)

Subtraction (a – b)

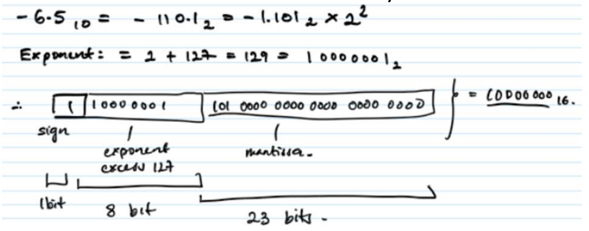
Convert b to its complement and perform addition.

Excess

*Binary rep. of x in excess-N = Binary rep. of (x + N)
*Dec rep. of x represented in excess-N: Convert x back to decimal and minus the result by N
** Excess-127 means the most negative number is 0-127 = -127. Since 8 bits, 2^8 – 127 = 129 non-neg nums.

IEE 754 Single-Precision Floating-Point Rep

1-bit sign, 8-bit exponent (**EXCESS 127)** 23-bit mantissa
Decimal to IEE: Convert to binary, normalize, fill parts
Normalized: Leftmost mantissa bit always 1.



Sign Extension

***Unsigned**: Place all bits* rightmost, fill unfilled front part w. 0.
***Signed**:* Copy **magnitude part**, place rightmost, copy sign bit and put it at first bit, fill the in -between gap with zeros
1s and 2s: Copy **all the bits** and place at rightmost part, fill the unfilled front part with the sign bit of the original num.

MIPS (Immediate is a **2s complement**)
Given k-bit addresses, you can have 2^k addresses.

Maximising Opcodes: Reserve one opcode for the smallest length fields and give the rest to the longer.
Minimising Opcodes: Reserve one opcode for the longest length field and give the rest to the smallest.

Branching

The immediate field = number of instructions skipped, starting from **end of the branch instruction**.
(if jump backwards **then immediate is -ve**, must count the branch instruction as one of the instruction to skip too!)
If branch is not taken: $PC_{new} = PC + 4$
If branch is taken: $PC_{new} = (PC + 4) + (immediate \times 4)$

Bitwise Logical Operators:

Getting NOT from NOR: nor x, x, $zero (x is the same register)
Getting NOT from XOR: xor x, x, y (y is register contain all 1s)

ANDI, ORI, XORI will **zero extend** 16-bit immediate **to the left**
LB and **LH** will **sign extend** 8 bits and 16 bits fetched respectively from the memory and store it in the register

SB and **SH** will only store the 8 and 16 **least significant bits** in the register mentioned respectively into the memory

LUI shifts the given 16 bit immediate to the left by 16 bits and concatenate it with 16 zeros.

To generate a mask (i.e. keep specific bits) using ANDI, use bit 1 to keep the bit you want and 0 for unwanted bit.
OR can be used to force certain bits to become 1.

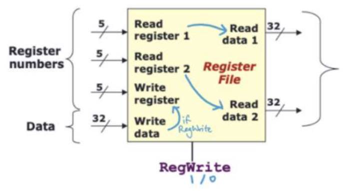
ISA (MSB = most significant Byte)
Big-endian: MSB are stored in lower addresses (MIPS)
Little-endian: LSB are stored in lower addresses

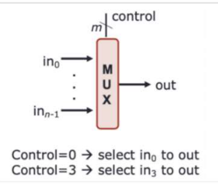
Data Path

Instruction Execution Cycle (5): Fetch, Decode, Operand Fetch, Execute, Result Write (Store)
Multiplexer: Inputs: n lines, Control: m bits where $n = 2^m$
Control: (Note the following control signals aren't complete)
RegDst: 0 -> choose rt; 1-> choose rd
ALUSrc: 0 -> choose RD2; 1-> choose SignExt(Inst[15:0])
MemToReg: 0 -> choose ALU result; 1-> choose Memory Read Data
PCSrc: 0 -> PC + 4; 1 -> (PC+4) + SignExt(Inst[15:0]) << 2

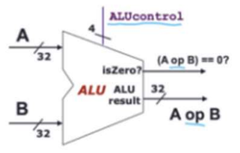
ALUOp
00 lw/sw | 01 beq | 10 R-Type

Register File, Multiplexer:

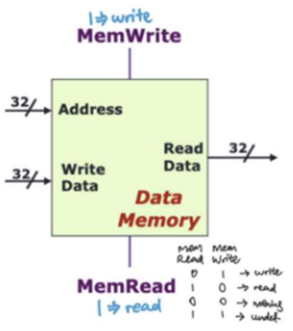



control
m
in$_0$
in$_{n-1}$
MUX
out
Control=0 → select in$_0$ to out
Control=3 → select in$_3$ to out

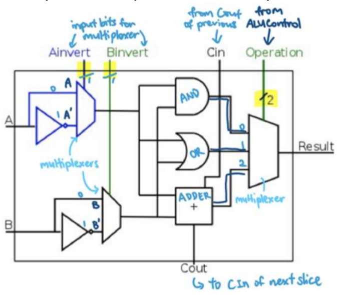
ALU, Data Memory:



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



ALU Slice: Data IN (*A, B, Cin*), Control IN (*A/B invert, Op)*



Parity:
An odd parity: total number of bits that have a value of 1 in a signal, including parity bit, must be an odd number. **Even parity** is opposite: the total number of "1" bits in a signal must be a positive number.
Diminished Radix Complement ((r-1)'s Complement):
Number N in base-r having n digits the (r-1)'s complement or Diminished Radix complement is defined as : *(r^n - 1) - N*
Radix Complement (aka r's Complement):
Given number N, base-r, n digits, defined as: *(r^n – N).*
This is also the same as adding 1 to (r-1)'s complement to get r's complement.

Control Signals:
These can be generated using opcode directly.

- **RegDst** @ Decode/Operand Fetch
  0/1: write register = Inst[20:16] / Inst[15:11]
- **RegWrite** @ Decode/Operand Fetch
  0/1: No register write / WD written to WR
- **ALUSrc** @ ALU (determines first input)
  0: Operand2 = Register Read Data 2
  1: Operand2 = SignExt(Inst[15:0]) (sign ext immediate)
- **MemRead** @ Memory
  0/1: no read / reads memory using Addr (returnd in RD)
- **MemWrite** @ Memory
  0/1: no write / writes Register RD 2 into mem[Address]
- **MemToReg** @ RegWrite
  0/1 register write data = ALU result / mem read data
- **PCSrc** @ Memory/RegWrite
  0/1: next PC = PC+4 or PC = SignExt(Inst[15:0])<< 2+(PC+4)
  PCSrc = set to 1 if Branch AND is0 are both 1, aka (isBranchInstruction AND branchIsTaken)

Memory in MIPS:
Each instruction is 32 bits aka 4 bytes. Each memory address Is one byte. Hence an instruction = 4 steps (a word). Each byte address is 32 bits. Hence: **0x4000 means 0x 0000 4000.**

Prefixes: 0: octal (8). 0x: hexa (16). 0b: binary (2)

Operation	Opcode in MIPS	Meaning
Addition	`add $rd, $rs, $rt`	`$rd = $rs + $rt`
	`addi $rt, $rs, C16`$_{2s}$	`$rt = $rs + C16`$_{2s}$
Subtraction	`sub $rd, $rs, $rt`	`$rd = $rs - $rt`
Shift left logical	`sll $rd, $rt, C5`	`$rd = $rt << C5`
Shift right logical	`srl $rd, $rt, C5`	`$rd = $rt >> C5`
AND bitwise	`and $rd, $rs, $rt`	`$rd = $rs & $rt`
	`andi $rt, $rs, C16`	`$rt = $rs & C16`
OR bitwise	`or $rd, $rs, $rt`	`$rd = $rs | $rt`
	`ori $rt, $rs, C16`	`$rt = $rs | C16`
NOR bitwise	`nor $rd, $rs, $rt`	`$rd = $rs ↓ $rt`
XOR bitwise	`xor $rd, $rs, $rt`	`$rd = $rs ^ $rt`
	`xori $rt, $rs, C16`	`$rt = $rs ^ C16`

`C5` is [0 to 2^5-1] `C16`$_{2s}$ is [-2^{15} to 2^{15}-1] `C16` is a 16-bit pattern

Non-critical Instruction Execution paths:
To generate control signals: I-MEM > Control (e.g. need ALUSrc signal). As long as Control latency lesser than Reg.File access latency, it will not be in critical path.
Calculate next PC: [I-MEM > Control > AND > MUX(PCSrc)] which is again not critical to the overall latency.

R Format:

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

each field is a 5/6-bit **unsigned** integer
opcode always = 0, shamt set to 0 for all non-shift instructions
rs set to 0 for sll/srl

I Format:

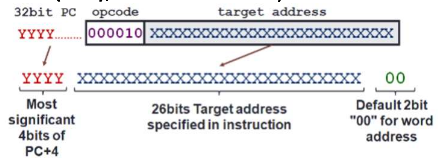
6	5	5	16
opcode	rs	rt	immediate

immediate is a **signed integer 2s complement** (up to 2^{16} values)

J Format:

6 bits	26 bits
opcode	target address

Jump
Given address of instruction, **drop the first 4 bits (MSB) and the last 2 bits (LSB),** the rest will be your immediate field.


32bit PC
opcode
target address
YYYY.........
000010
XXXXXXXXXXXXXXXXXXXXXXXXXX
YYYY
XXXXXXXXXXXXXXXXXXXXXXXXXX
00
Most significant 4bits of PC+4
26bits Target address specified in instruction
Default 2bit "00" for word address

Instruction Execution:
Single-cycle implementation: time taken depends on slowest instruction. Clock cycle same time as slowest instruction.
Multi-cycle implementation: time taken depends on number of steps. Cycle time is determined by slowest step.

Cycle Time: (Given component resource latencies), consider **propagation delay** for instruction processing aka **critical** (longest) path. Sum latencies. Disregard faster parallel paths.

SUB (R-type) Critical path (6): I-Mem > Reg.File > MUX(ALUSrc) > ALU > MUX(MemToReg) > Reg.File

LW Critical Path (6): I-Mem > Reg.File > ALU > DataMem > MUX(MemToReg) > Reg.File