Lecture 24

Assembly Logic Instructions

- AND is useful for masking bits
- OR is useful for combining bit fields in 2 registers

Shift Instructions

- shift instructions
  - shift left logical
  - shift right logical
  - shift right arithmetic (shift most significant bit with sign bit)

```
# s5 = 1111 1111 0001 1110 0010 0000 1011 0111
slli t0, s5, 7
# t0 = 1000 1111 0001 0000 0101 1011 1000 0000
srli s1, s5, 17
# s1 = 0000 0000 0000 0000 0111 1111 1000 1111
srai t2, s5, 3
# t2 = 1111 1111 1111 1111 1111 1111 1000 1111
```

- shifting left by N is equivalent to multiplying by 2^N
- shifting right by N is equivalnt to dividing by 2^N, where sign comes into play

Example -32 = 11100000, division by 8 gives 11111100 = -4 which does not work with logical shift

- Logical shifts used with and/or can extract or assemble bit fields

```
# s7 = 0x1234ABCD
srli s6, s7, 8 # s6 = 0x001234AB
andi s6, s6, 0xFF # s6 = 0x000000AB
```


- Logical shifts used with and/or can extract or assemble bit fields

```
# s7 = 0x1234ABCD
srli s6, s7, 8 # s6 = 0x001234AB
andi s6, s6, 0xFF # s6 = 0x000000AB
```


- Accessing a byte or half-word of memory
  - E.g. char
  - `lb`, `lbu`: load byte (unsigned), where byte is etended with zeros or the sign
  - `lh`, `lhu`: same but half-word (2 bytes)
  - `sb`, `sh`: stores least significant byte/half of reg to memory; only changes 2 bytes of memory

E.g.

address	3	2	1	0
data	E1	80	16	01

```
addi s4, zero, 0 # s4 = 0x00000000
lbu s1, 3(s4) # s1 = 0x000000E1
lb s2, 3(s4) # s2 = 0xFFFFFFE1
lh s3, 0(s4) # s3 = 0x00001601
lhu s5, 0(s4) # s5 = 0xFFFF1601
sb s2, 0(s4) # updates mem at address 0 to be E1
```

Note that s4 is the address/pointer to the data. Halfword addresses must be evenly divisible by 2, and word addresses must be evenly divisible by 4.

Program Flow

- Each is 4 bytes long
- Consecutive instruction addresses incrase by 4

Memory Address	Instruction
0x538	addi s1, s2, s3
0x53C	lw t2, 8(s1)
0x540	sw, s3, 4(t6)

- Branch Instructions
  - We want a program to perform different tasks based on input, e.g. if/else, while
  - branch instructions are used to modify program flow, so instructions do not have to be executed in sequential order
  - Conditional branch: preform iff
  - Many flavours, e.g. for comparing 2 source registers, beq (if), bne (!if), blt (signed <), bltu (unsigned <)
  - Pseudo instructions, e.g. beqz, bnez, where z stands for zero