Exam 2 is Live!

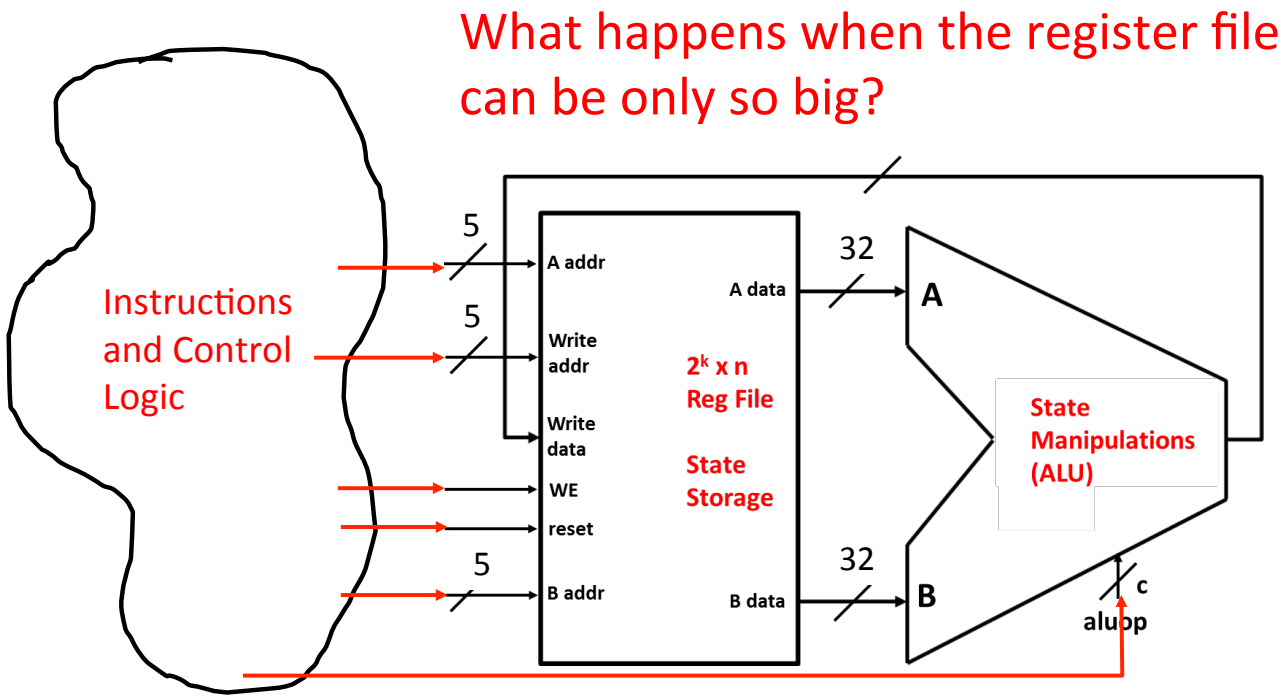
MIPS Load & Stores

1 handout

Today's lecture

- **MIPS Load & Stores**
  - Data Memory
  - Load and Store Instructions
  - Encoding
  - How are they implemented?

State – the central concept of computing



We need more space!

Registers

- Fast
- High Bandwidth
- Synchronous

- Small (32x32 bits)
- Expensive

Main Memory

- Slow
- Low Bandwidth
- Not always synchronous

- Large (2^{32}B)
- Cheap-ish

Harvard Architecture stores programs and data in *separate* memories

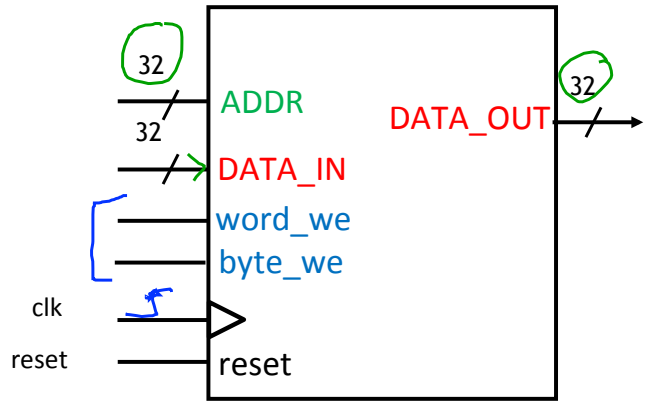
Instruction memory:

- Contains instructions to execute
- Treat as read-only

Data memory:

- Contains the data of the program
- Can be read/written

Data Memory is byte-addressable with 2^{32} bytes



word = 32b = 4B

word_we	byte_we	Operation	
0	0	Read (Load)	DATA_OUT = M[ADDR]
0	1	Write (Store) byte in ADDR	M[ADDR] = DATA_IN[7:0]
1	0	Write (Store) word in ADDR	M[ADDR]† = DATA_IN[31:0]

MEMORY

32b

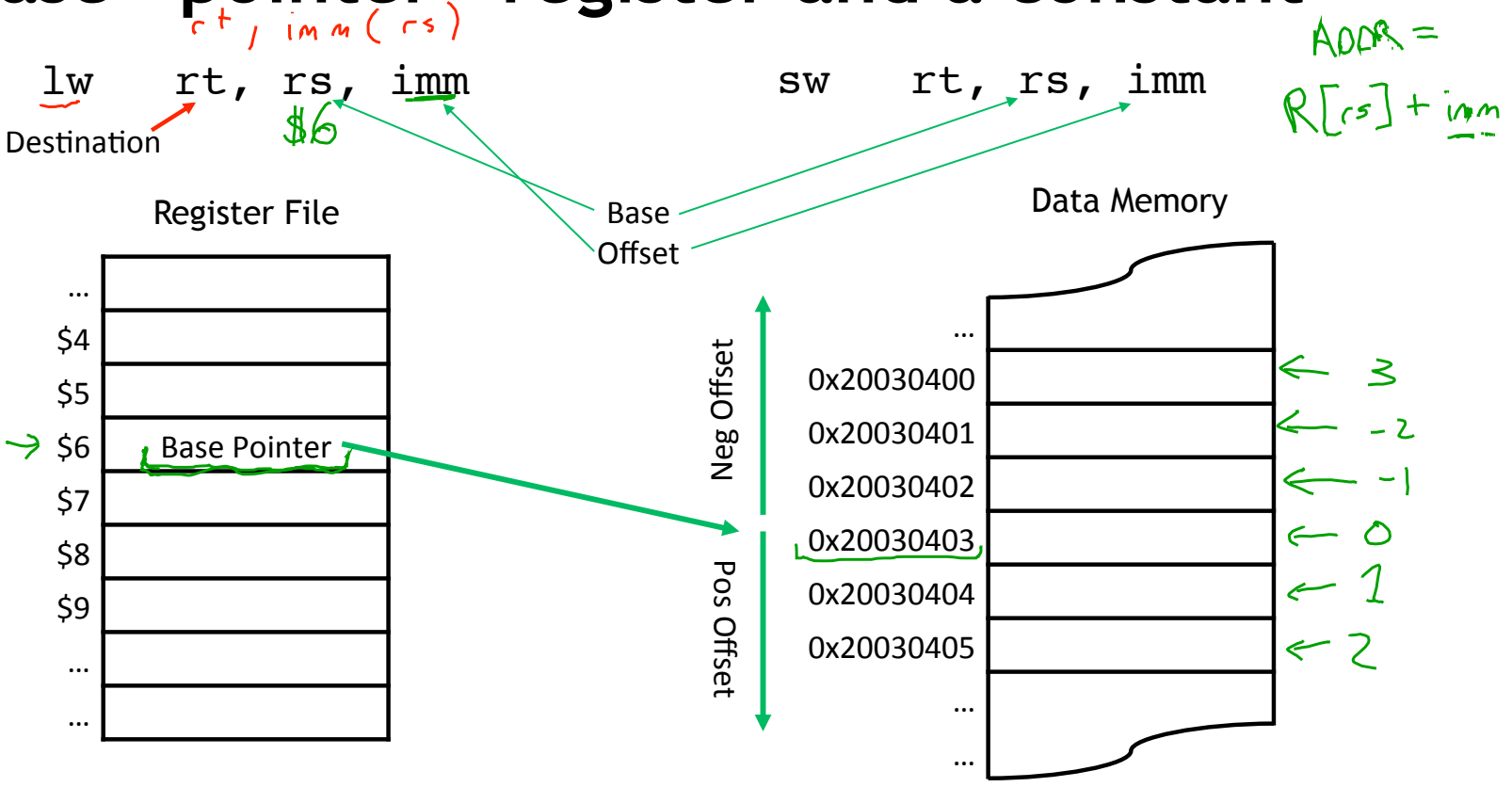
synch

†(ADDR must be word aligned; e.g., ADDR[1:0] == 2'b00)

We can load or store bytes or words

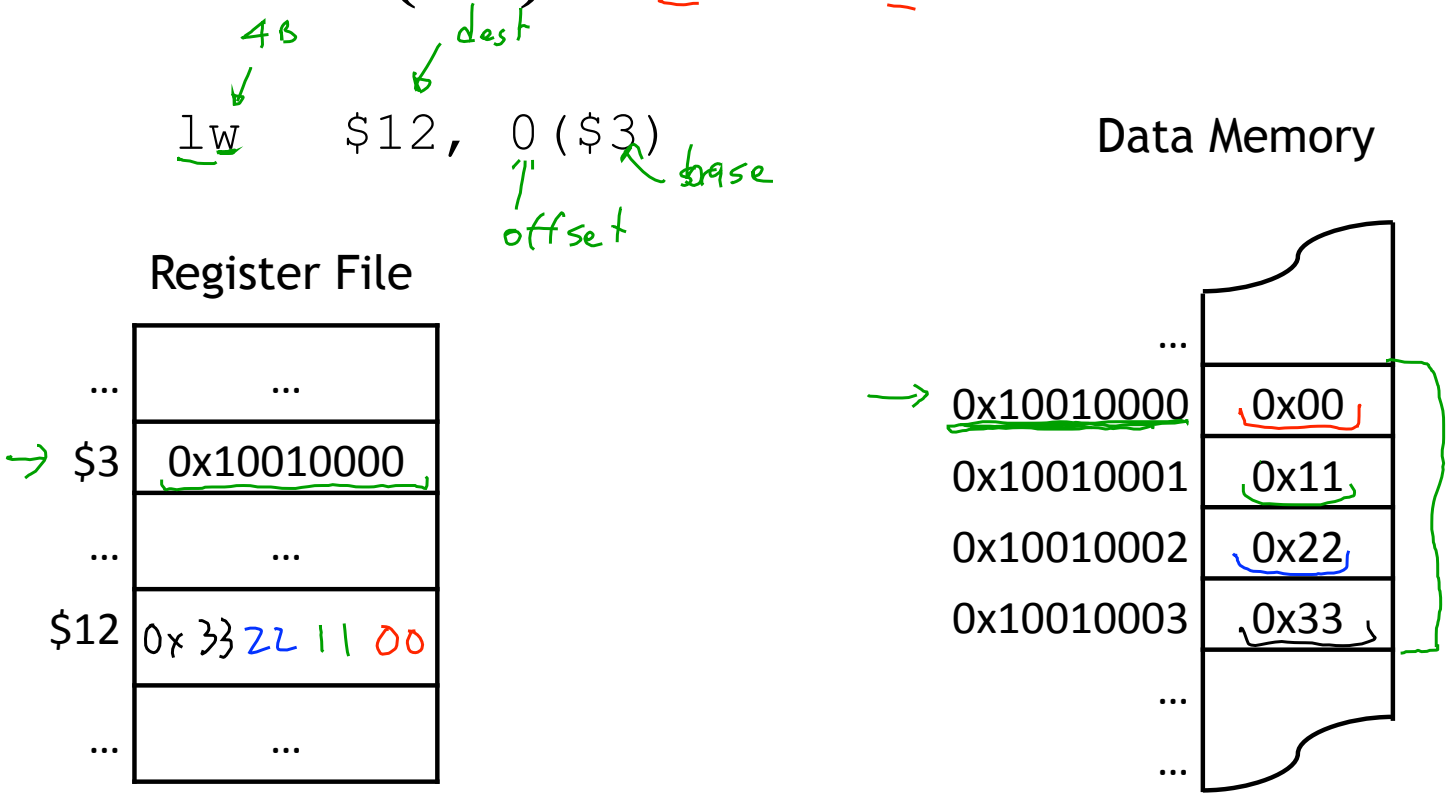
	Load	Store
Word	**lw** R[rt] = M[ADDR][31:0]	**sw** M[ADDR] = R[rt][31:0]
Byte	**lb** R[rt] = SEXT(M[ADDR][7:0]) **lbu** R[rt] = ZEXT(M[ADDR][7:0])	**sb** M[ADDR] = R[rt][7:0]

Indexed addressing derives ADDR from a base "pointer" register and a constant



Load word (lw) is Little Endian

```
lw    $12, 0($3)
```



i>clicker

```
sw    $12, 0($3)
```

Value src

rt

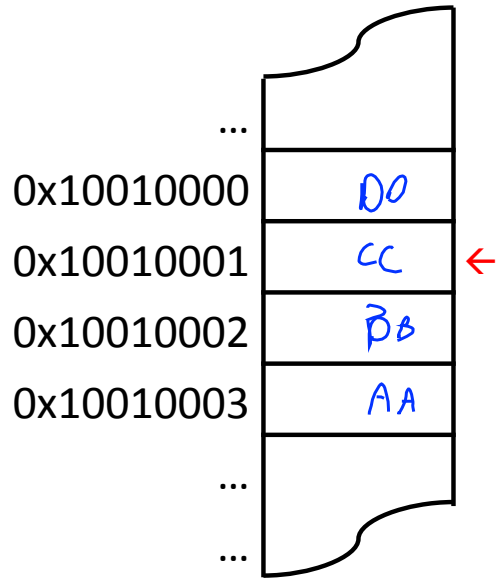
imm

rs (base)

Register File

...	...
$3	0x10010000
...	...
$12	0xAABBCCDD
...	...

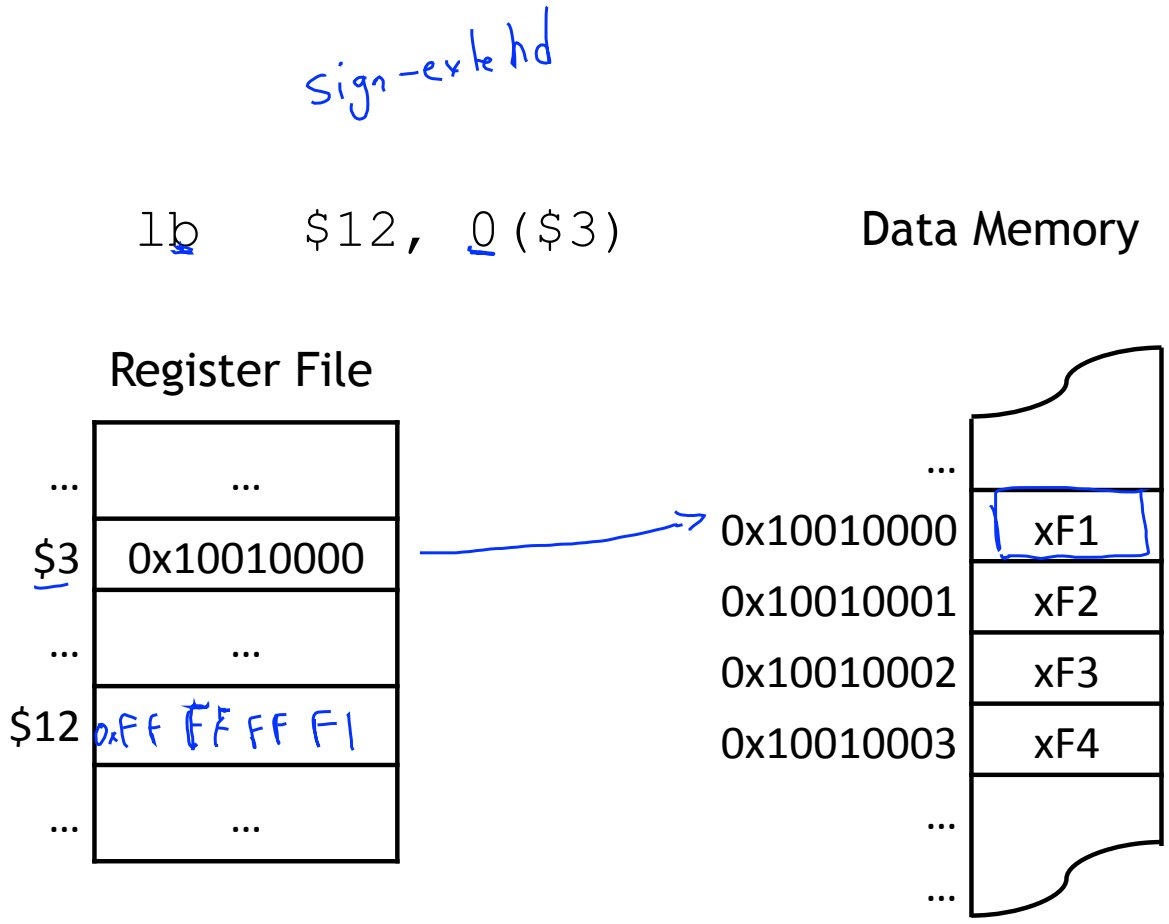
Data Memory



a) 0xAA
b) 0xBB
c) 0xCC
d) 0xDD

sign-extend

```
lb    $12, 0($3)
```



i>clicker

R[3] + 1

```
lbu   $12, 1($3)
```

offset

Register File

…	…
$3	0x10010000
…	…
$12	
…	…

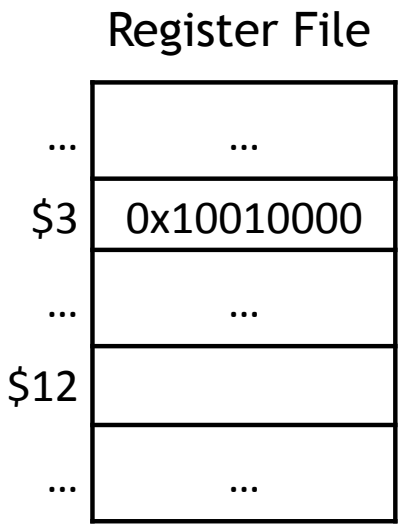
Data Memory

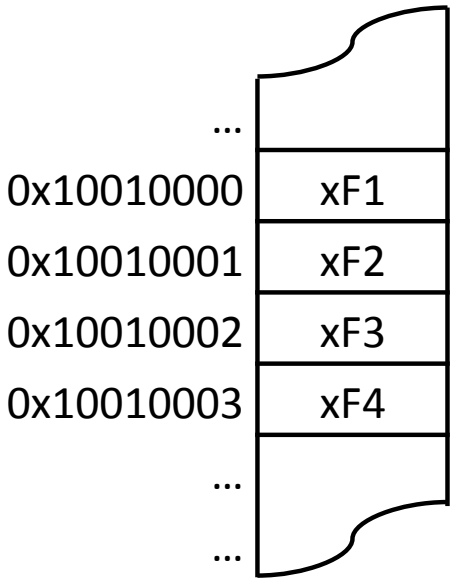
…	
R3 → 0x10010000	a) ←
R3 + 1 → 0x10010001	b) ←
0x10010002	c) ←
0x10010003	
0x10010004	d) ←
0x10010005	e) ←
0x10010006	
0x10010007	
0X10010008	
…	
…	

Where is data read from?

i>clicker

```
lbu   $12, 1($3)
```





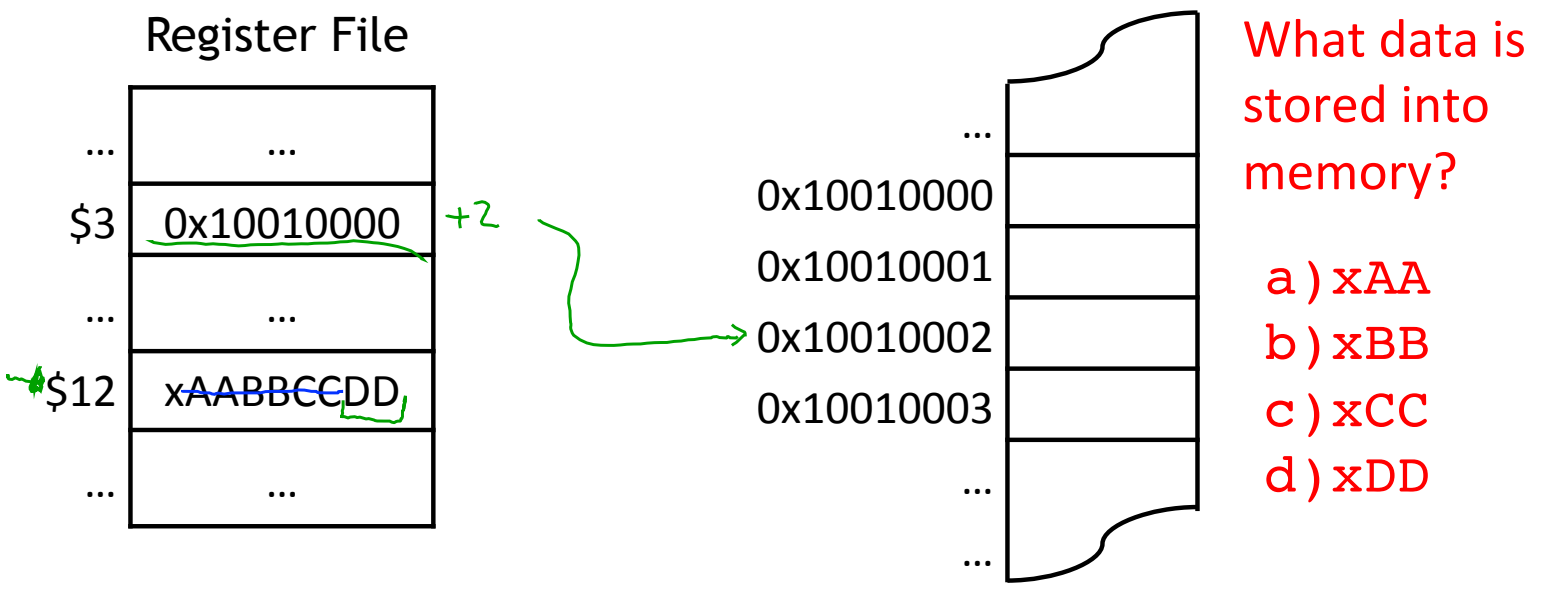
What data is loaded into the register?

a) 0x0000F200
b) 0x000000F2
c) 0xFFFFF200
d) 0xFFFFFFF2

i>clicker

value src

```
sb   $12, 2($3)
```



What data is stored into memory?

a) xAA
b) xBB
c) xCC
d) xDD

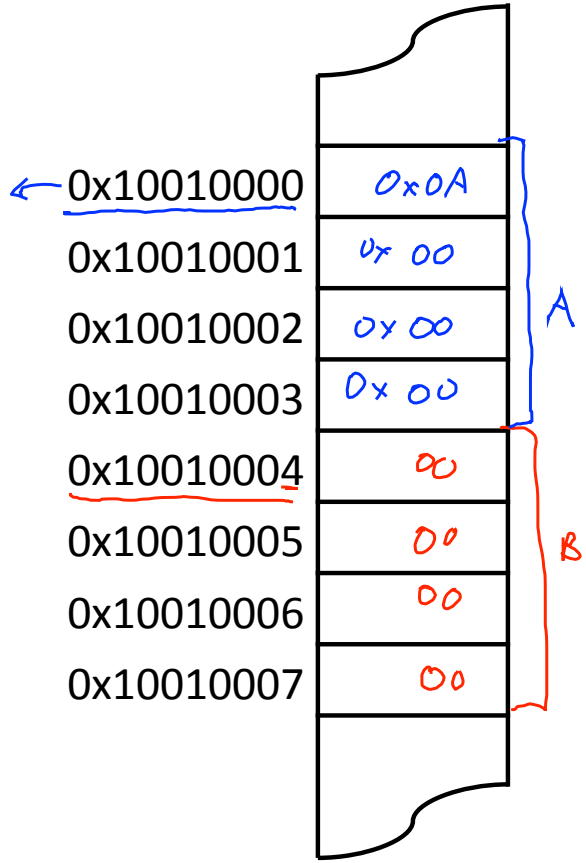
Convert C code into MIPS assembly

```
int a = 10;
int b = 0;
void main() {
   b = a+7;
}
```

Convert C code into MIPS assembly

```
int a = 10;
int b = 0;
void main() {
    b = a+7;
}
```

```
.data
a: .word 10
b: .word 0
.text
main:
    la   $4, a
```



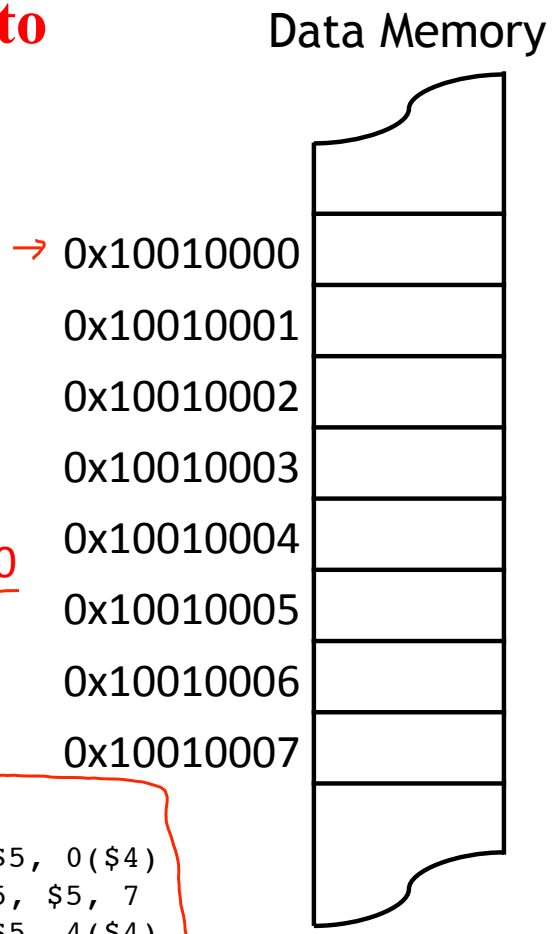
Which code finishes converting C code into MIPS assembly?

i-clicker

```
int a = 10;
int b = 0;
void main() {
    b = a+7;
}
```

```
.data
a: .word 10
b: .word 0
.text
main:
    la   $4, a
```

$4 = 0x10010000



A

```
lw    $5, 0($4)
addi $5, $5, 7
sw    $5, 0($4)
```

B

```
lw    $5, 0($4)
addi $5, $5, 7
sw    $5, 1($4)
```

C

```
lw    $5, 0($4)
addi $5, $5, 7
sw    $5, 2($4)
```

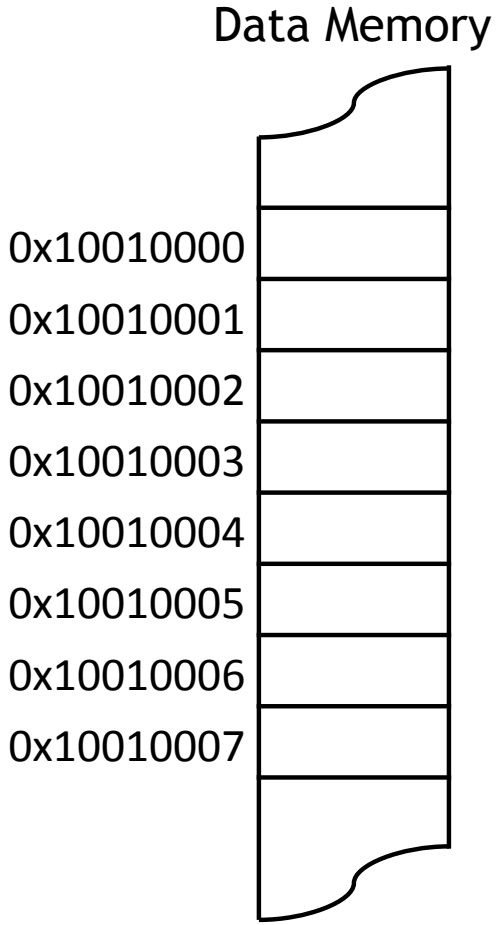
D

```
lw    $5, 0($4)
addi $5, $5, 7
sw    $5, 4($4)
```

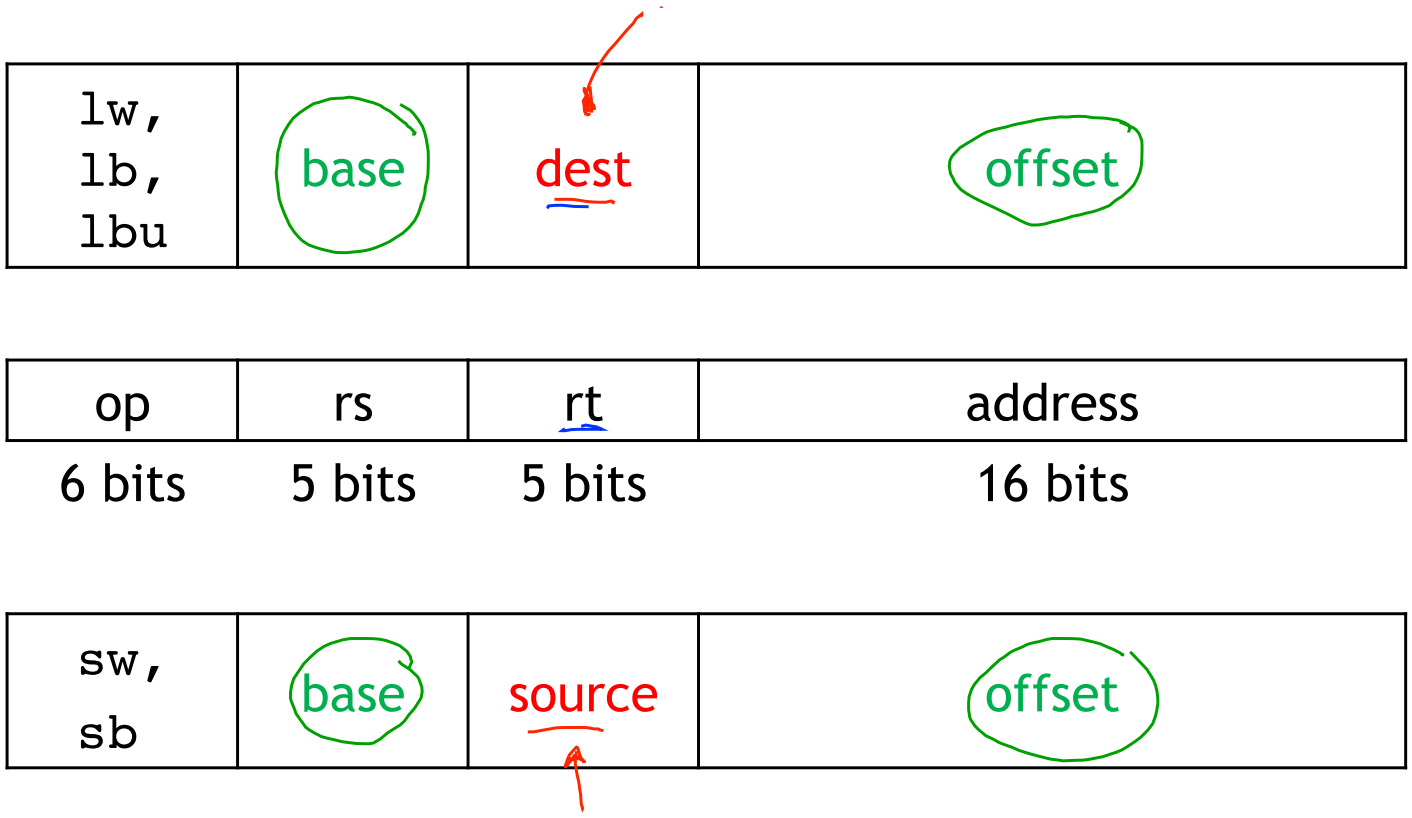
Convert C code into MIPS assembly

```
int a = 10;
int b = 0;
void main() {
   b = a+7;
}
```

```
.data
a: .word 10
b: .word 0
.text
main:
    la   $4, a
    lw   $5, 0($4)
    addi $5, $5, 7
    sw   $5, 4($4)
```

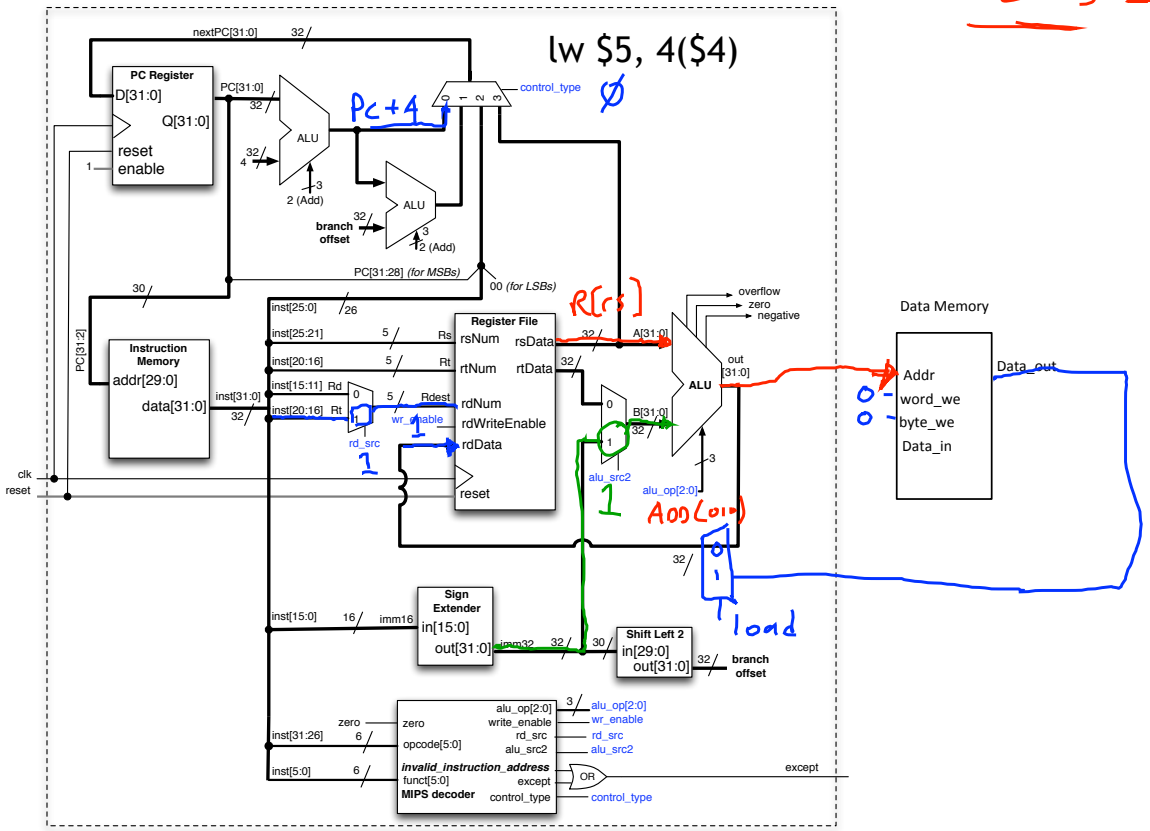


Loads and stores use the I-type format

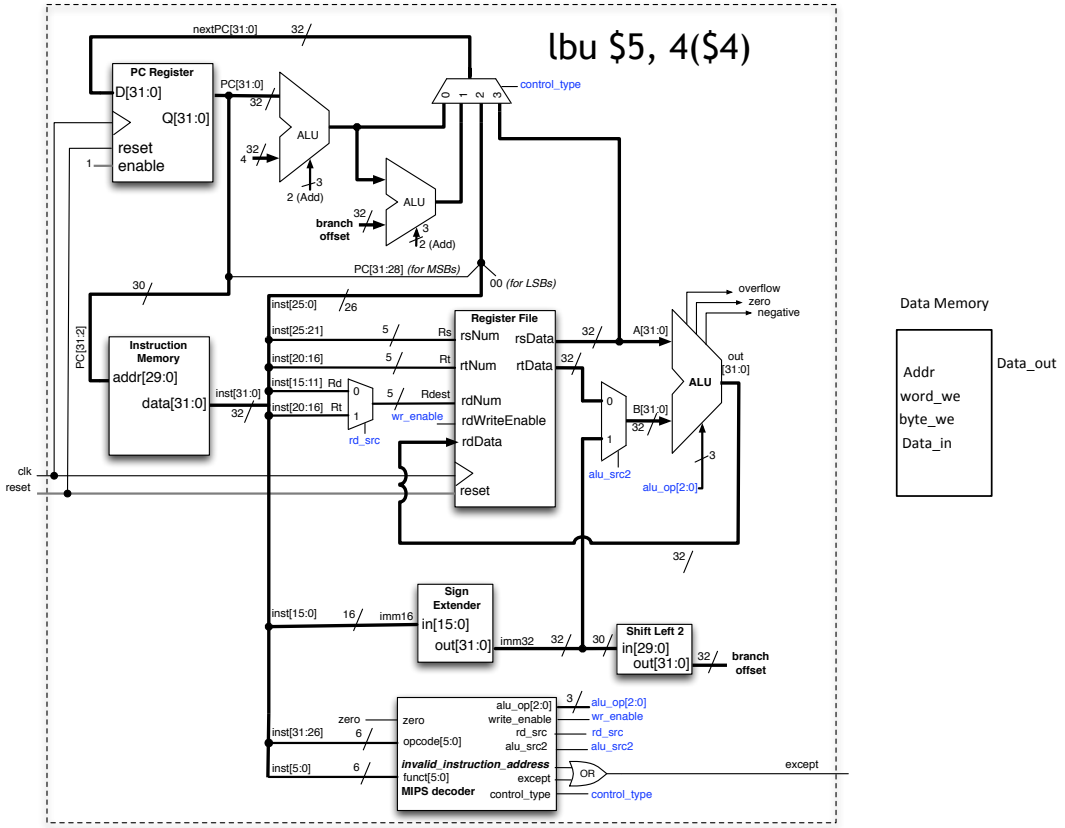


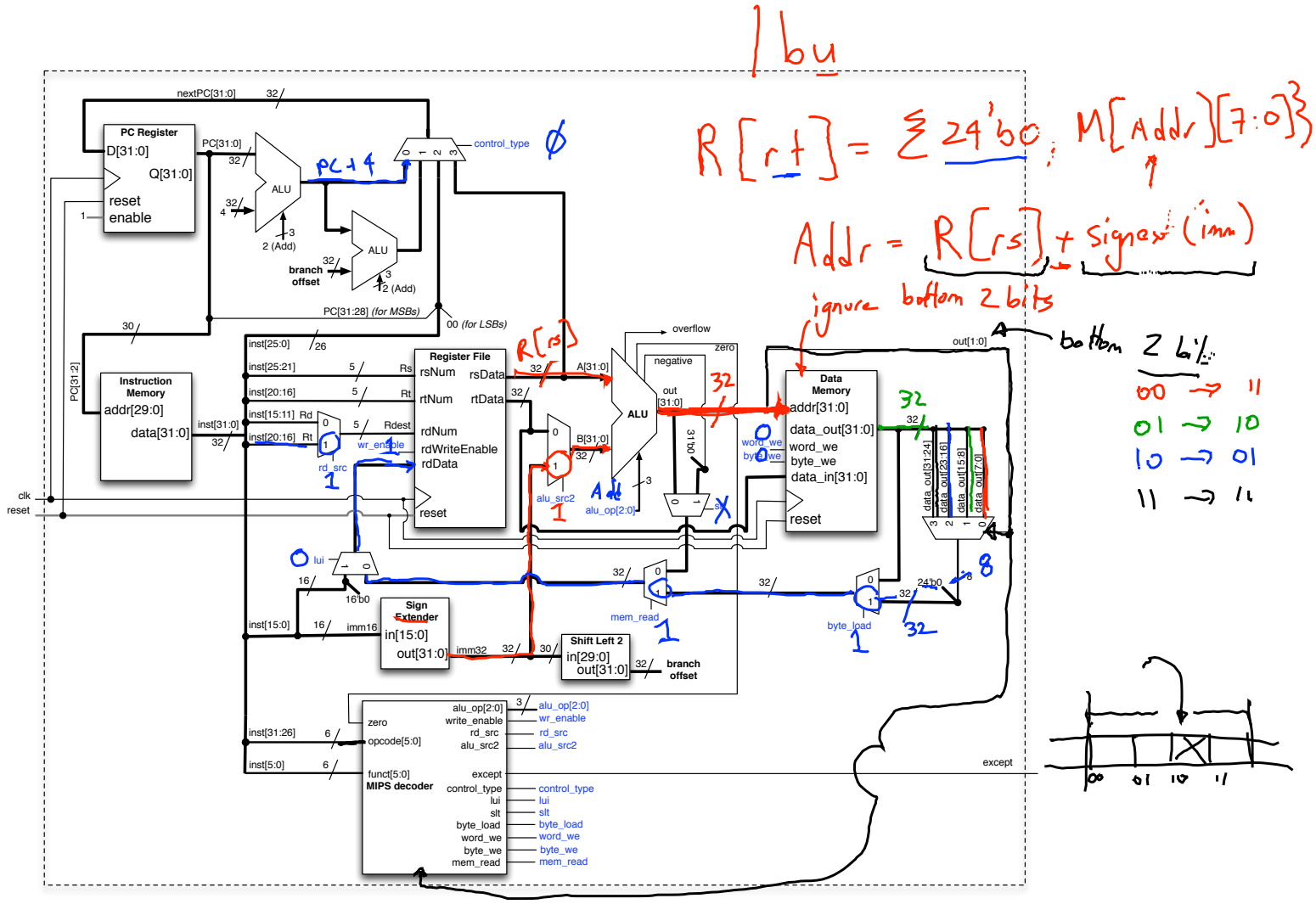
load word implemented

Addr = R[rs] + sign ext(imm)

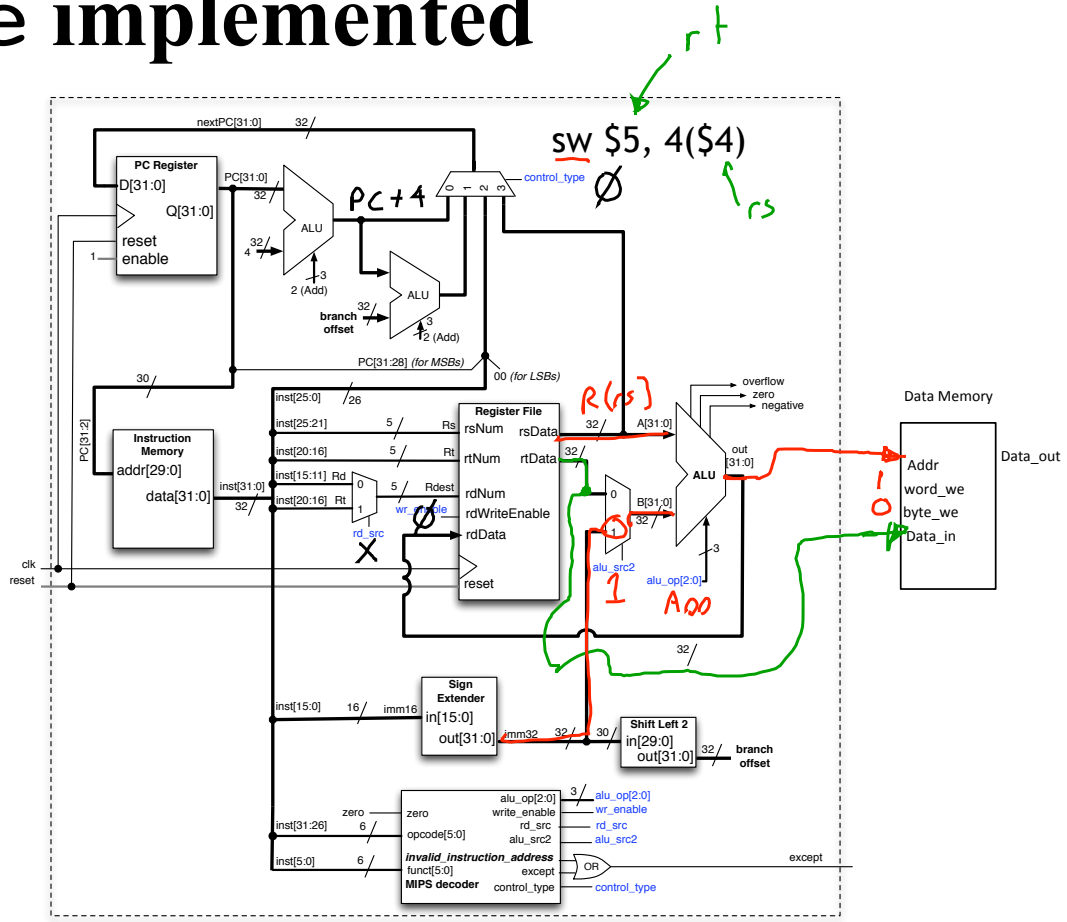


load byte unsigned **implemented**




lbu
R[rt] = { 24'b0 ; M[Addr][7:0] }
Addr = R[rs] + signex(imm)
ignore bottom 2 bits
bottom 2 bits
00 → 11
01 → 10
10 → 01
11 → 11
nextPC[31:0]
PC Register
D[31:0]
Q[31:0]
reset
enable
PC[31:0]
ALU
pc+4
control_type
branch offset
2 (Add)
PC[31:28] (for MSBs)
00 (for LSBs)
Instruction Memory
addr[29:0]
data[31:0]
inst[31:0]
inst[25:0]
inst[25:21]
inst[20:16]
inst[15:11] Rd
inst[20:16] Rt
Register File
rsNum
rsData
rtNum
rtData
rdNum
rdWriteEnable
rdData
Rs
Rt
Rdest
wr_enable
rd_src
R[rs]
A[31:0]
B[31:0]
alu_src2
alu_op[2:0]
overflow
zero
negative
out[31:0]
Data Memory
addr[31:0]
data_out[31:0]
word_we
byte_we
data_in[31:0]
out[1:0]
data_out[31:24]
data_out[23:16]
data_out[15:8]
data_out[7:0]
lui
16'b0
Sign Extender
in[15:0]
out[31:0]
imm16
imm32
mem_read
byte_load
24'b0
Shift Left 2
branch offset
clk
reset
MIPS decoder
zero
opcode[5:0]
funct[5:0]
except
write_enable
wr_enable
lui
slt
byte_load
word_we
byte_we
mem_read

`store` implemented



Full Machine Datapath – Lab 6