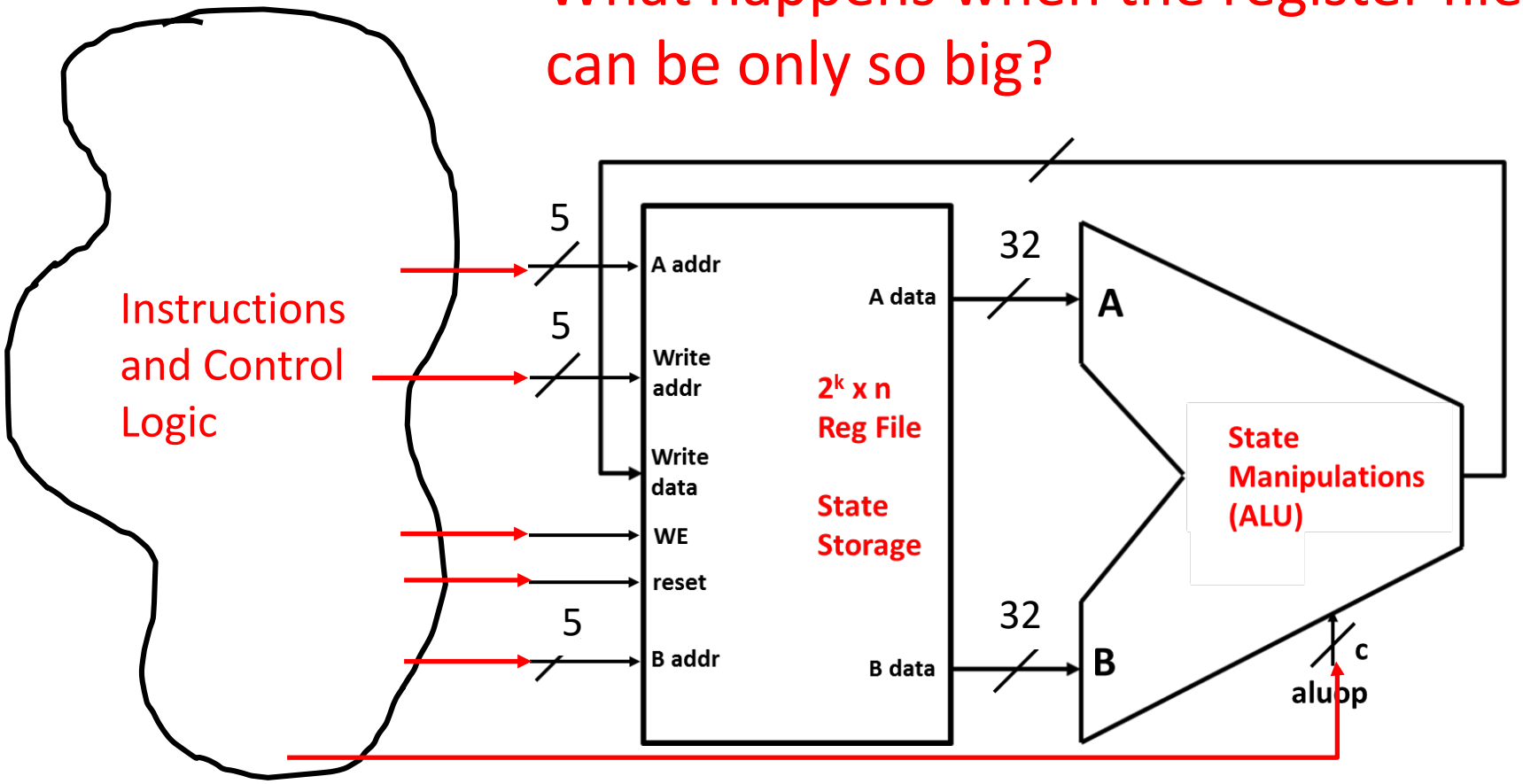
MIPS Load & Stores

Today's lecture

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

State – the central concept of computing

What happens when the register file can be only so big?



We need more space!

Registers	Main Memory
Fast	Slow
Synchronous	Not always synchronous
Small (32x32 bits)	Large (2^{32}B)
Expensive	Cheap-ish

Harvard Architecture stores programs and data in *separate* memories

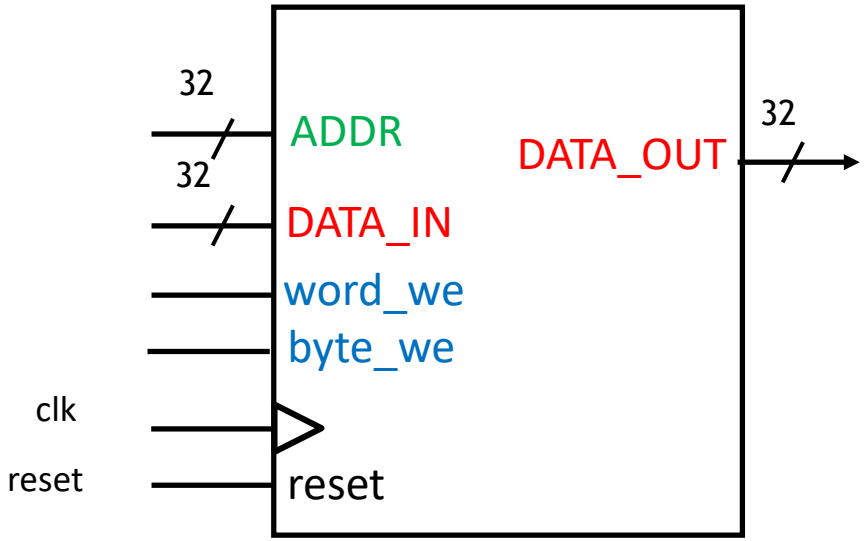
Instruction memory:

- 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_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]

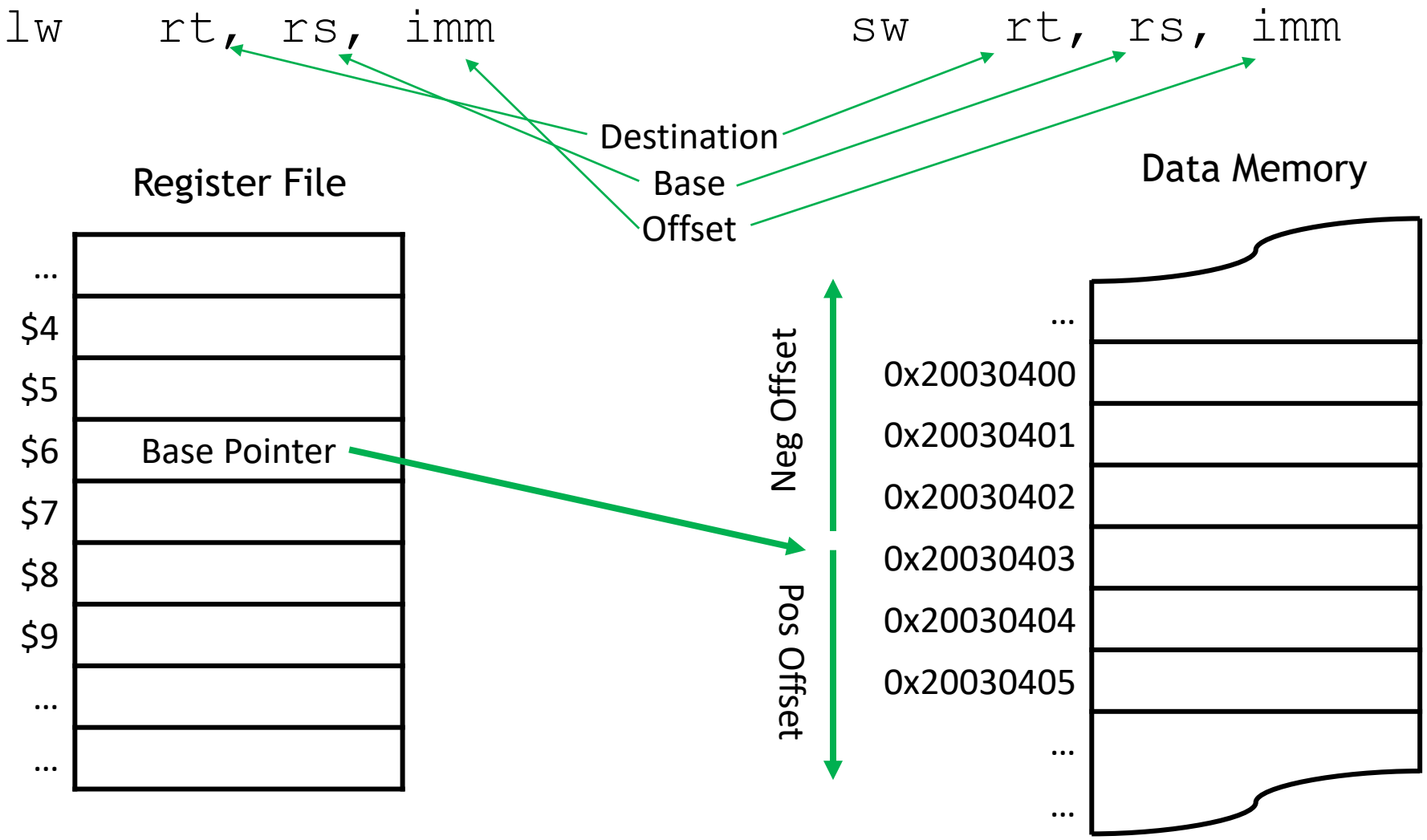
†(ADDR[1:0] must be word aligned)

We can load or store bytes or words

	Load	Store
Word	`lw` R[rt] = M[ADDR][31:0]	`sw` M[ADDR] = R[rs][31:0]

Byte	`lb`
	R[rt] = SEXT(M[ADDR][7:0])
	`lbu`
	R[rt] = ZEXT(M[ADDR][7:0])
		`sb`
		M[ADDR] = R[rs][7:0]

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



Load word (`lw`) is Little Endian

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

Register File

…	…
$3	0x10010000
…	…
$12	
…	…

Data Memory

…	
0x10010000	0x00
0x10010001	0x11
0x10010002	0x22
0x10010003	0x33
…	
…	

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

Register File

…	…
$3	0x10010000
…	…
$12	0xAABBCCDD
…	…

Data Memory

…	
0x10010000	
0x10010001	←
0x10010002	
0x10010003	
…	
…	

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

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

Register File

…	…
$3	0x10010000
…	…
$12	
…	…

Data Memory

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

Where is data read from?

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

Register File

…	…
$3	0x10010000
…	…
$12	
…	…

Data Memory

…	
0x10010000	xF1
0x10010001	xF2
0x10010002	xF3
0x10010003	xF4
…	
…	

What data is loaded into the register?

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

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

Register File

…	…
$3	0x10010000
…	…
$12	xAABBCCDD
…	…

Data Memory

…	
0x10010000	
0x10010001	
0x10010002	
0x10010003	
…	
…	

What data is stored into memory?

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

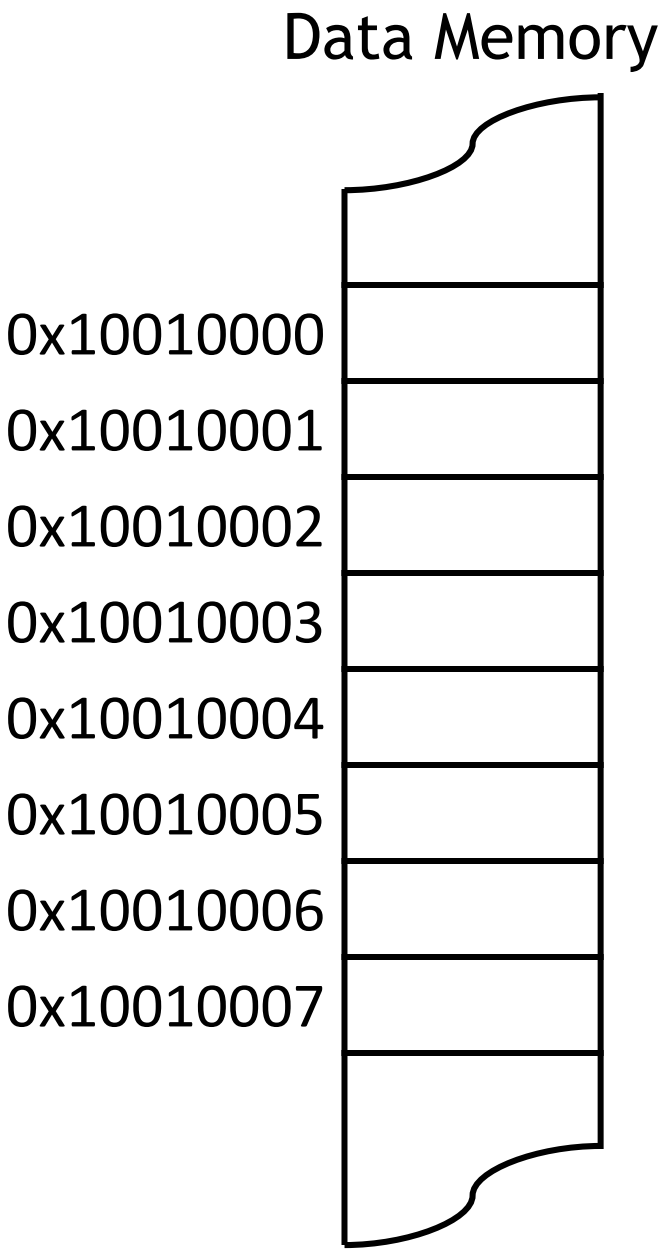
Convert C code into MIPS assembly

```
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
```



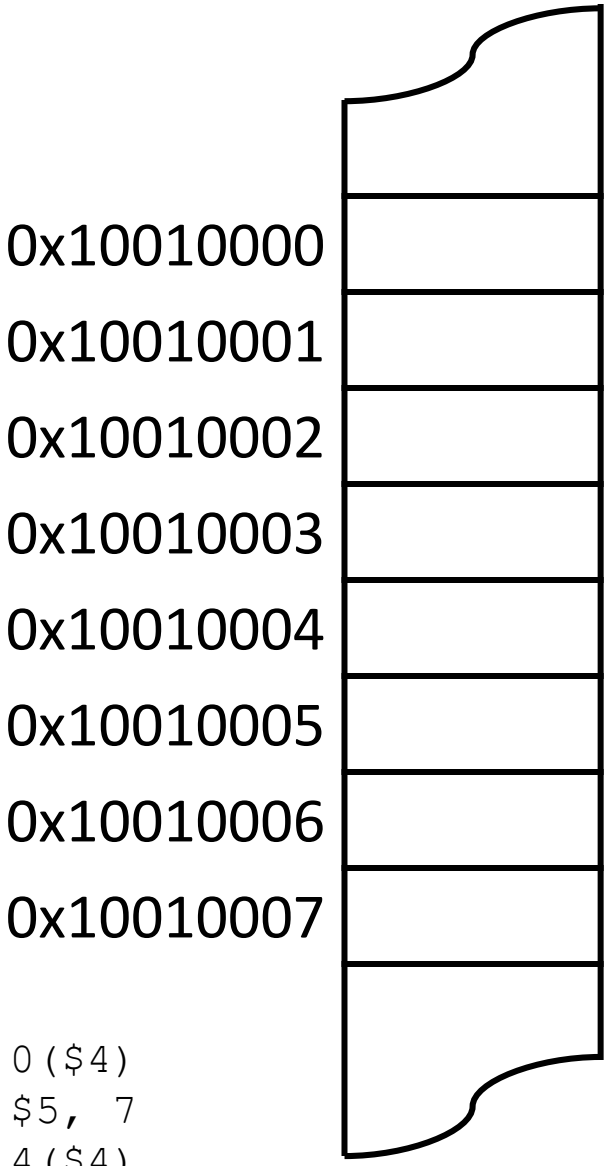
Which code finishes converting C code into MIPS assembly?

```
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

Data Memory



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)
```

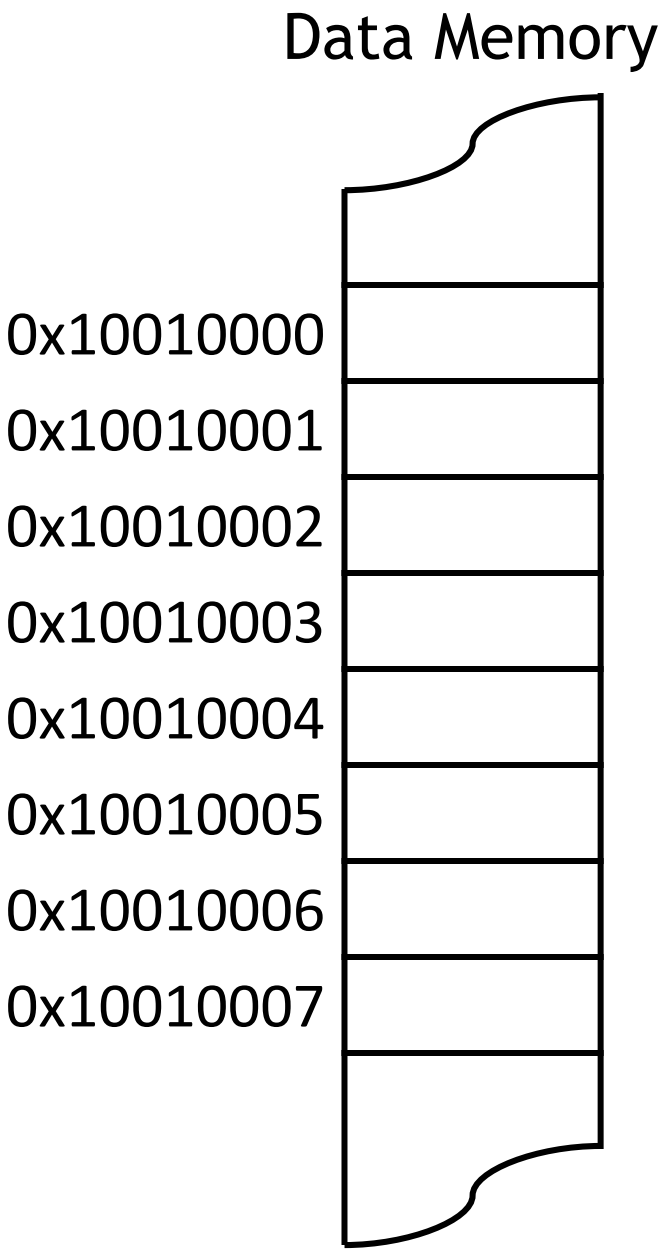
D

```
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

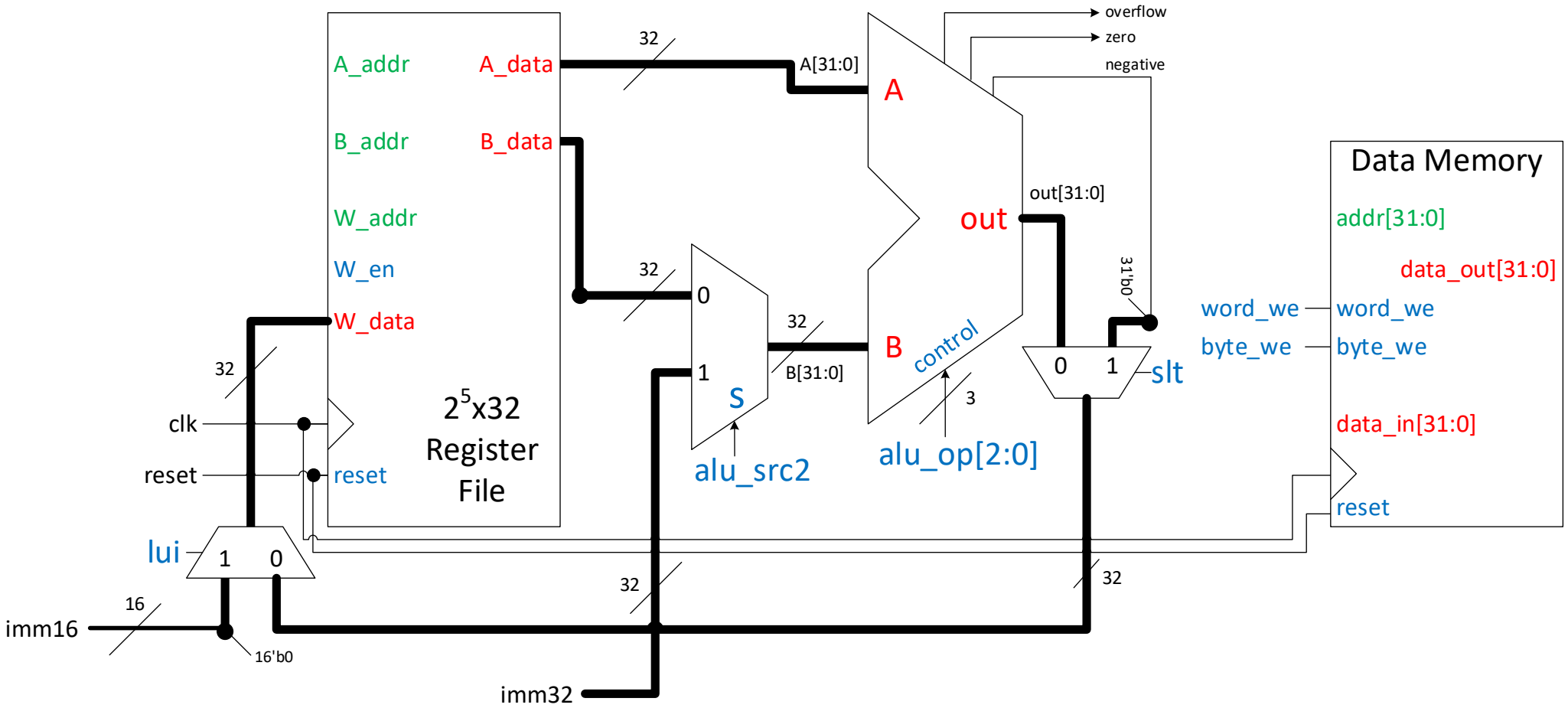
`lw`, `lb`, `lbu`	base	source	offset

op	rs	rt	address
6 bits	5 bits	5 bits	16 bits

`sw`, `sb`	base	dest	offset

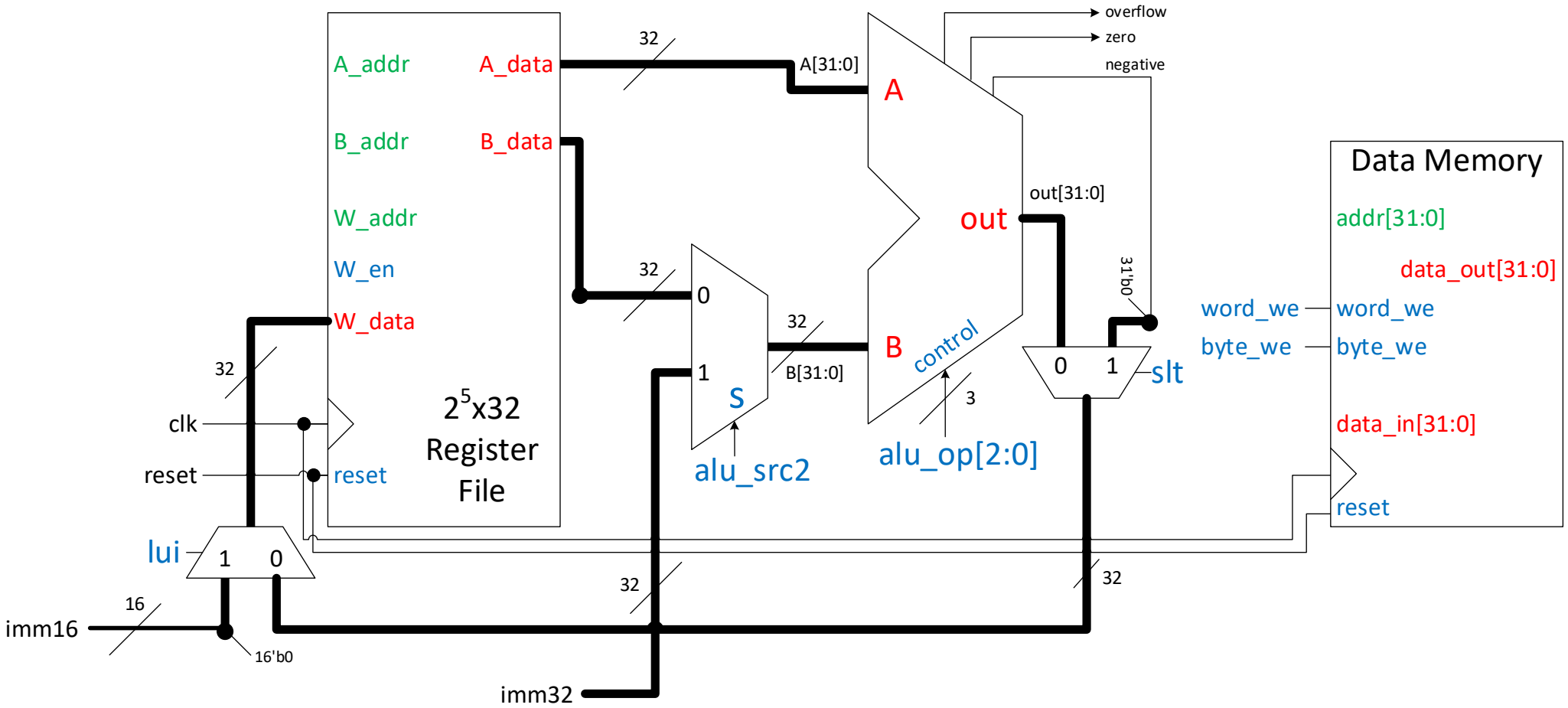
store implemented

sw $5, 4($4)



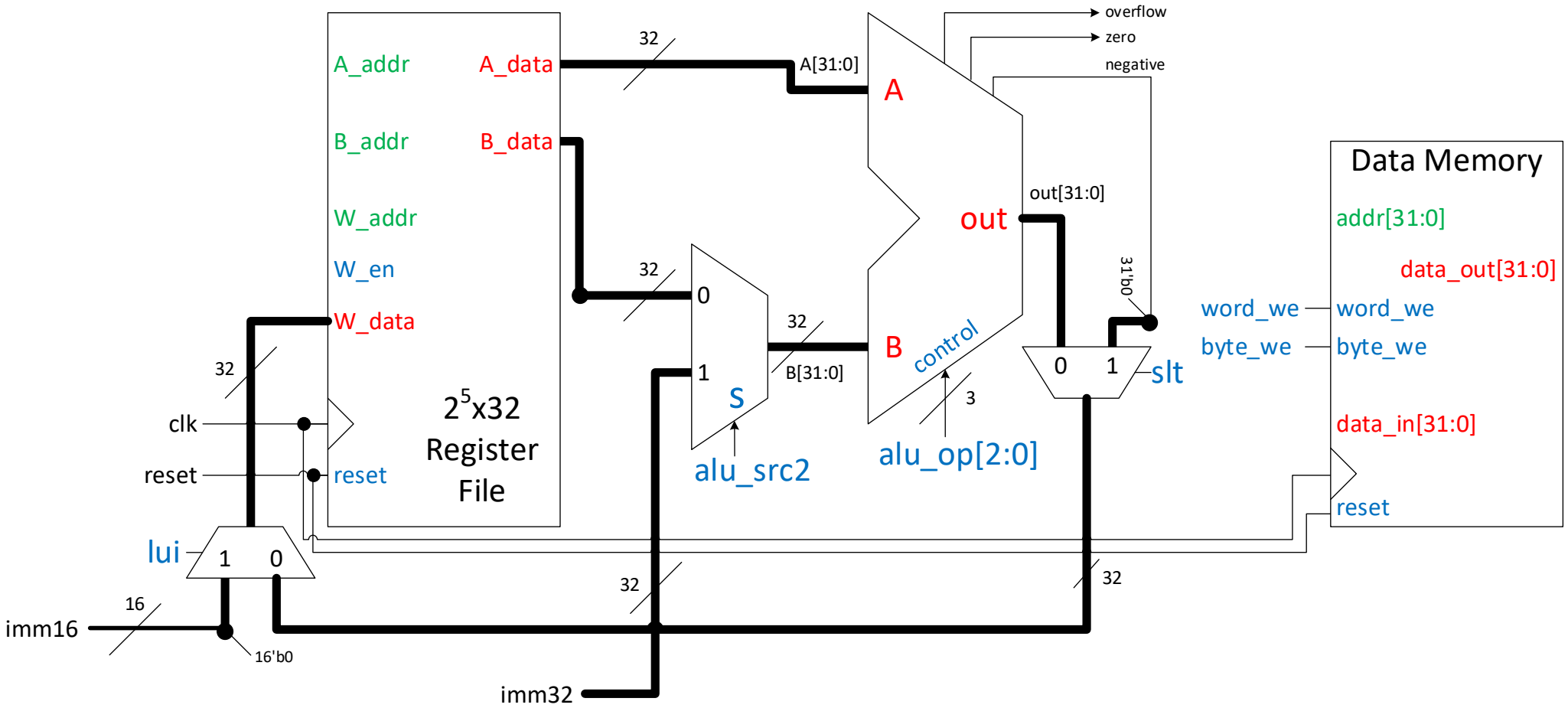
load word **implemented**

lw $5, 4($4)



load byte unsigned **implemented**

lbu $5, 4($4)



Full Machine Datapath – Lab 6