

---

# Computer Organization: A real processor

---

MIPS 2000

---

# Background

- We have built a model processor in ActiveHDL
  - You get to make it work
- Heavily based on MIPS2000
  - Described by Patterson & Hennessy
- Single-cycle design
  - All operations take 1 (long) cycle

---

# Instruction Set Specs

- 32 registers
- Load-Store Architecture
- Word Addressing
- 3 Formats for Instructions
  - Register to Register
  - Immediate
  - Jump

# Instruction Encodings

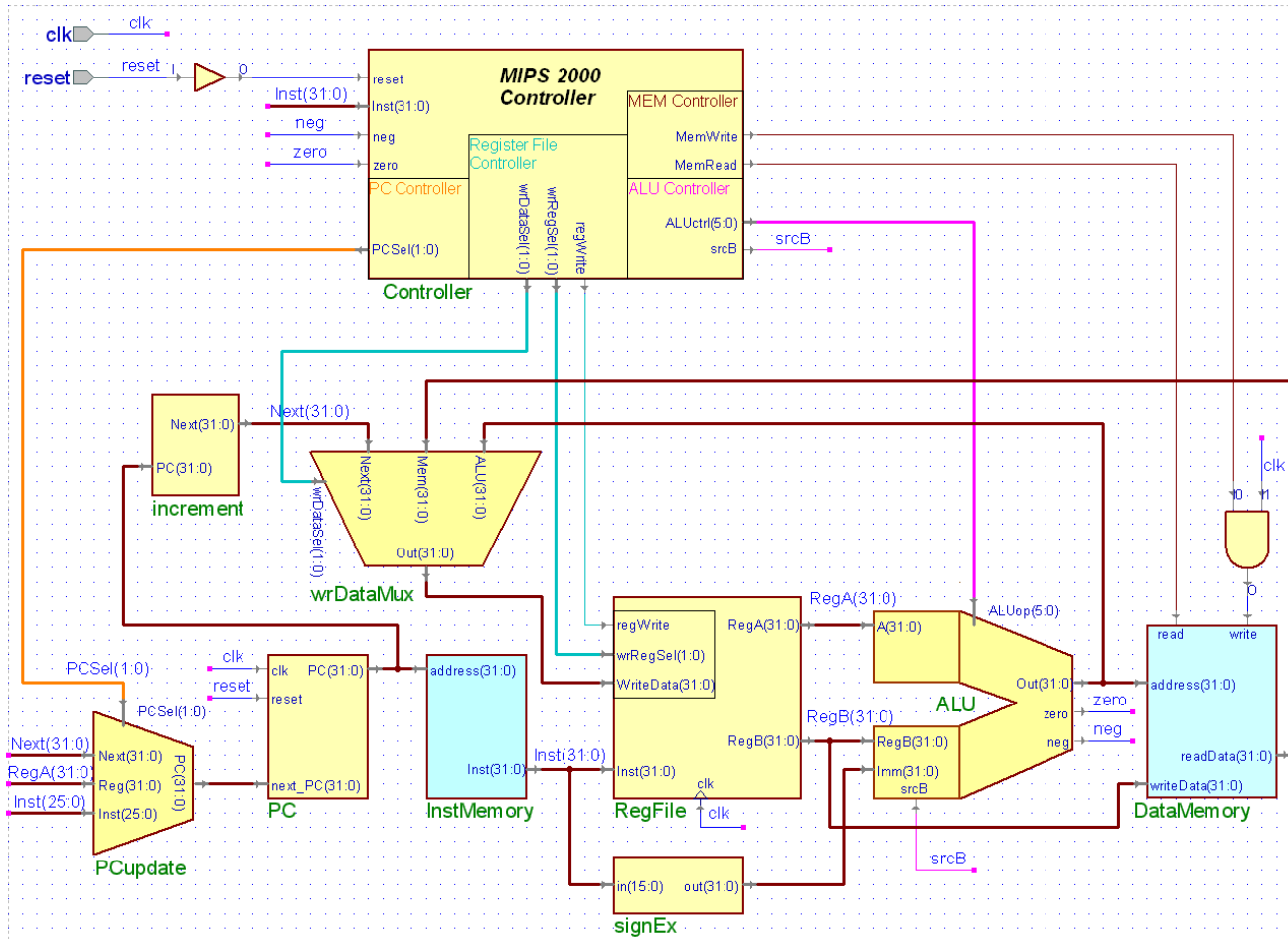
- Three principal types (32 bits in each instruction)

type	op	rs	rt	rd	shift	funct
R(egister)	6	5	5	5	5	6
I(mmediate)	6	5	5	16- immediate		
J(ump)	6	26- immediate				

- Some of the instructions

R	add	6'h00	rs	rt	rd	6'h20	rd = rs + rt
	sub	6'h01	rs	rt	rd	6'h22	rd = rs - rt
	and	6'h02	rs	rt	rd	6'h24	rd = rs & rt
	or	6'h03	rs	rt	rd	6'h25	rd = rs   rt
	slt	6'h04	rs	rt	rd	6'h2a	rd = (rs < rt)
I	lw	6'h23	rs	rt	offset		rt = mem[rs + offset]
	sw	6'h2b	rs	rt	offset		mem[rs + offset] = rt
	beq	6'h04	rs	rt	offset		pc = pc + offset, if (rs == rt)
	addi	6'h08	rs	rt	offset		rt = rs + offset
J	j	6'h02	target address				pc = target address
	halt	6'h3f	-				stop execution until reset

# Mips2000



# Program Counter

```
assign offset = {{16{Inst[15]}},Inst}; // sign extend  
the immediate
```

```
assign Branch = Next + offset;
```

```
assign Jump = {Next[31:26],Inst[25:0]}; // used by  
J instruction
```

```
// There are 4 possible sources for PC
```

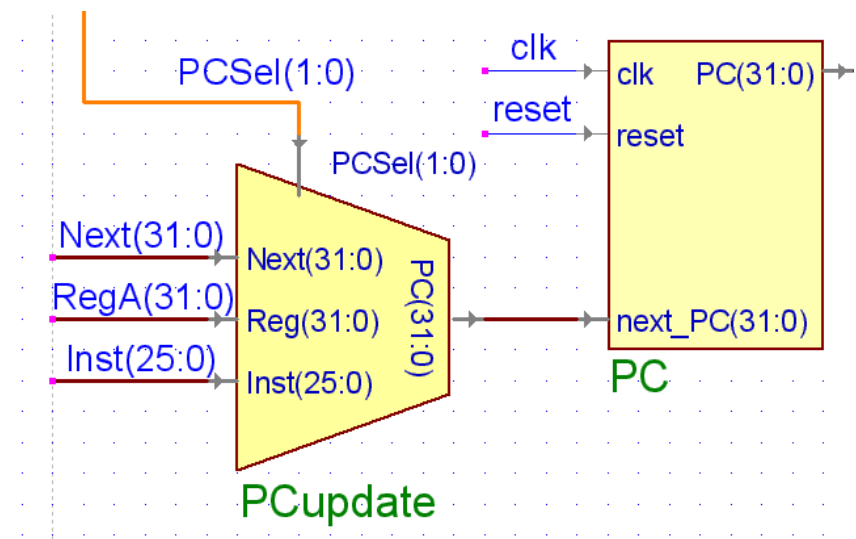
```
// 0. PC = Next (Move to next Instruction)
```

```
// 1. PC = Next + offset (Conditional Branch)
```

```
// 2. PC = Reg (Jump to Register value)
```

```
// 3. PC = Next[31:26],jump_target( J  
instruction)
```

```
assign PC = (PCSel[1])?  
((PCSel[0])? Jump : Reg) :  
((PCSel[0])? Branch : Next);
```



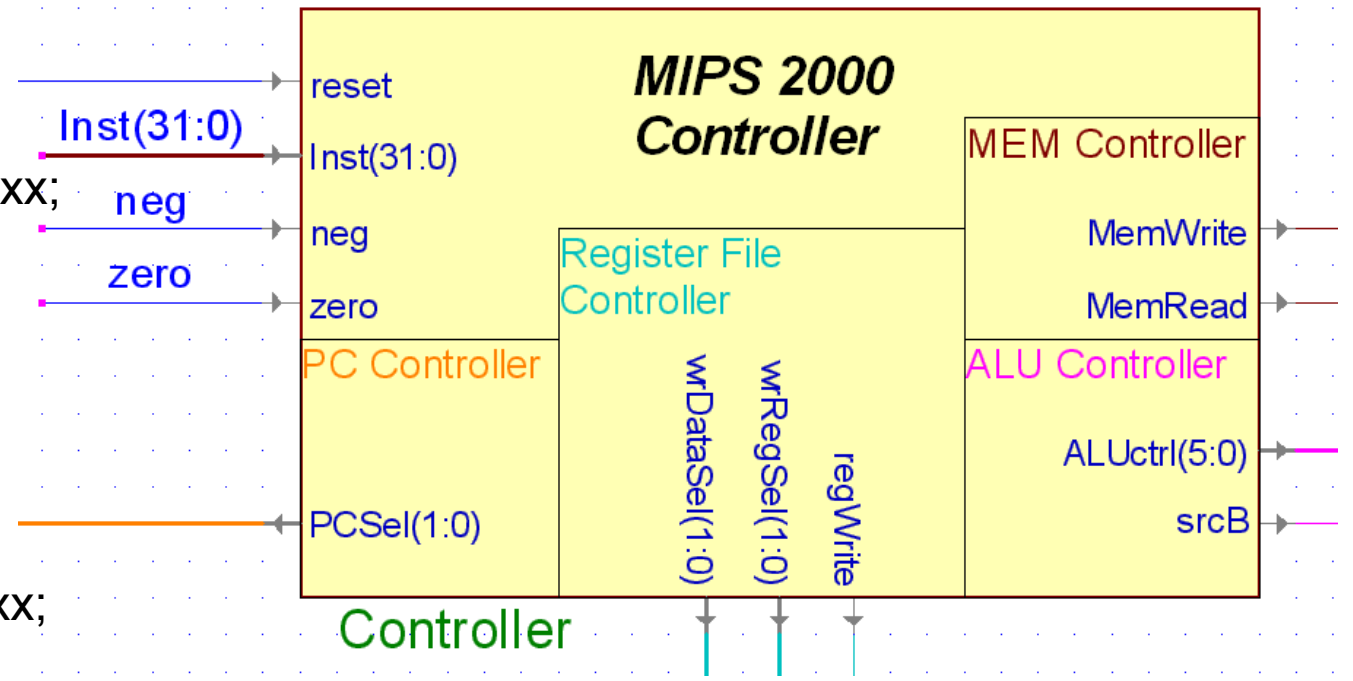
# Controller

## Skeleton Code:

...

```

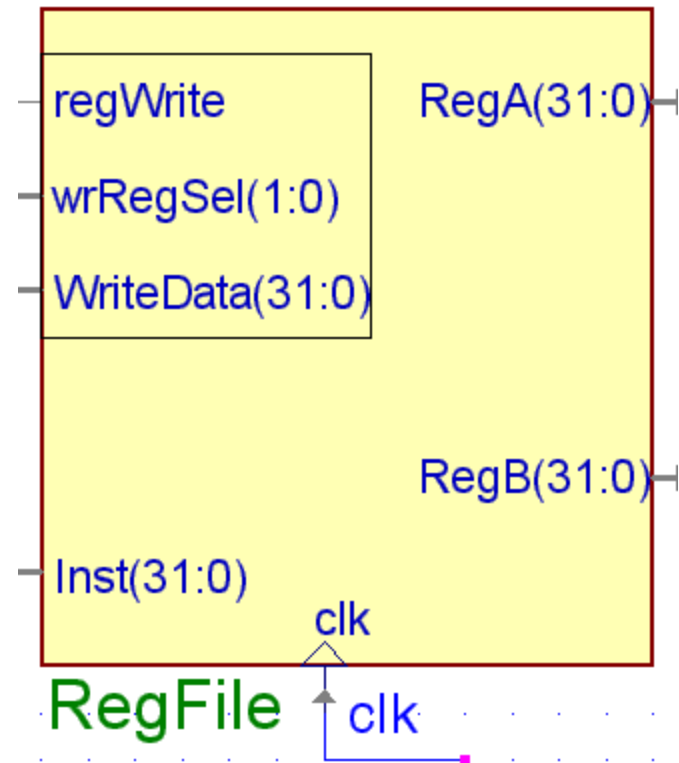
ADDI: begin
  wrDataSel = 2'bx;
  mw = 1'bx;
  mr = 1'bx;
  PCSel = 2'bx;
  srcB = 1'bx;
  regWrite = 1'bx;
  wrRegSel = 2'bx;
  op = 6'bxxxxxx;
end
  
```



# Register File

```
// decide which register is the one that might be
// written to (depends on instruction)
// 00 – rd, 01 – rt, 1X – hardwired to 31 for JAL
assign wrReg = wrRegSel[1] ?
    5'b11111 : (wrRegSel[0] ? rd : rt);
// do two reads and, optionally, one write with the
// register file
// read two registers and send them to the ALU
assign RegA = RegFile[rs];
assign RegB = RegFile[rt];

// write into a register (but not the register storing
// our constant 0)
always @(posedge clk) begin
    if (regWrite && (wrReg != 0)) begin
        RegFile[wrReg] = WriteData;
    end
end
```

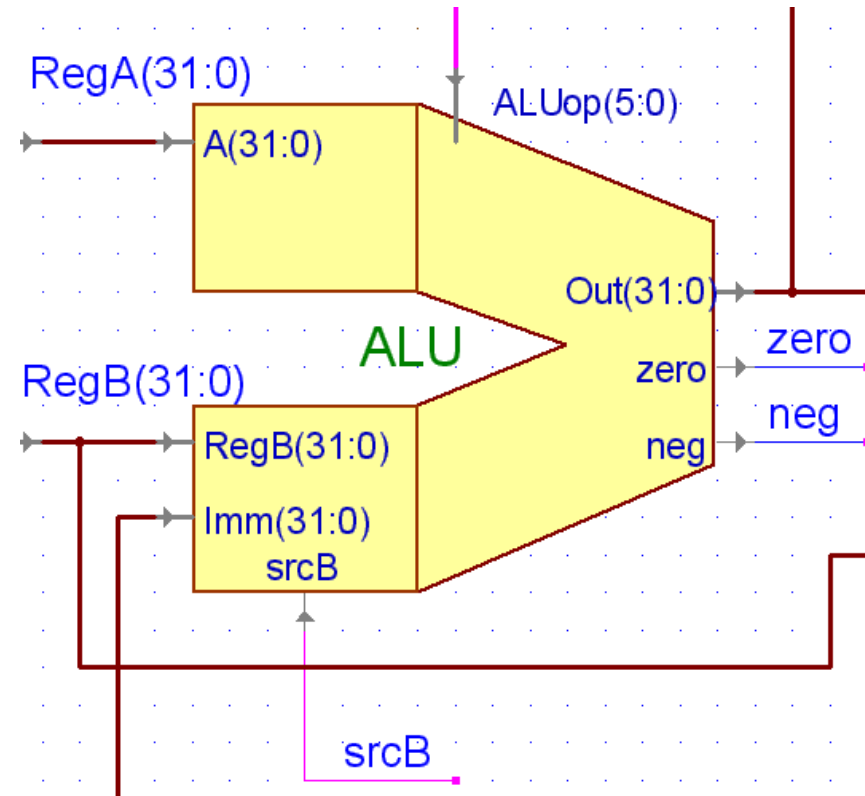




# ALU

```
// use srcB to select between RegB
// and Imm
assign B = (srcB === 0)? RegB :
Imm;

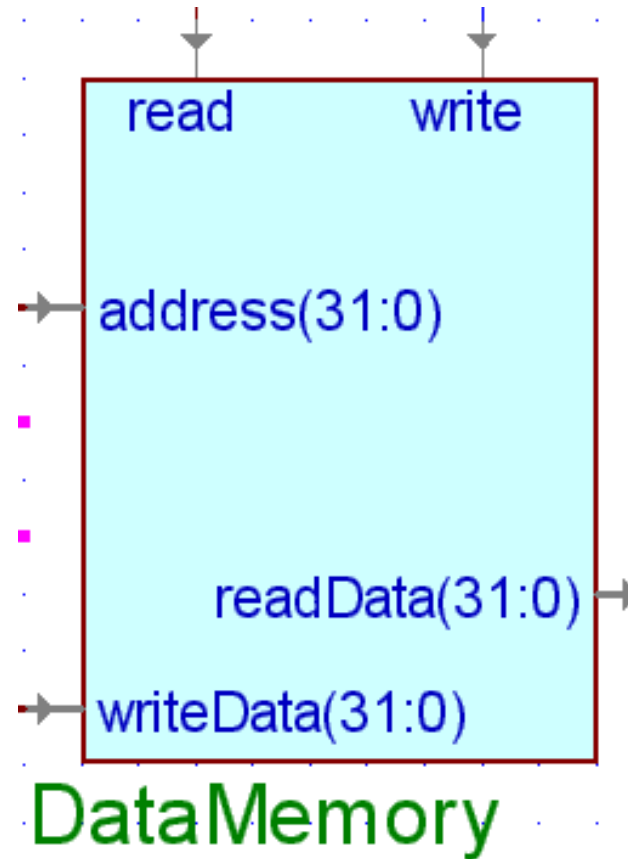
always @(A or B or op) begin
  case (op)
    6'b000001: result = A + B;
    ...
    default: result = 32'hxxxxxxxx;
  endcase
  zero = (result == 32'h00000000);
  neg = result[31];
end
```



# Data Memory

- Address from ALU
- Data from Reg B
- Memory-Mapped I/O
  - SW to xFFFFFFFF
  - Buffers / Displays

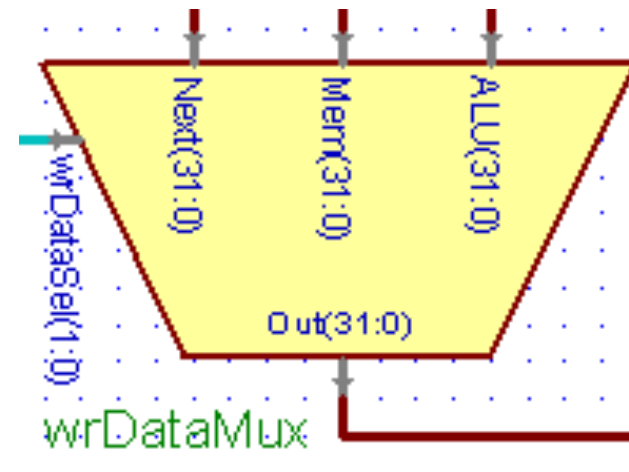
(See dmemory.v for more)



# Miscellaneous

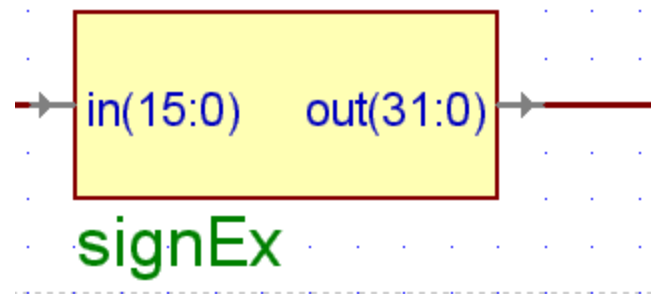
## WrRegSel:

```
// wrDataSel  
// 00: Out = ALU  
// 01: Out = MEM  
// 1X: Out = PC + 1
```

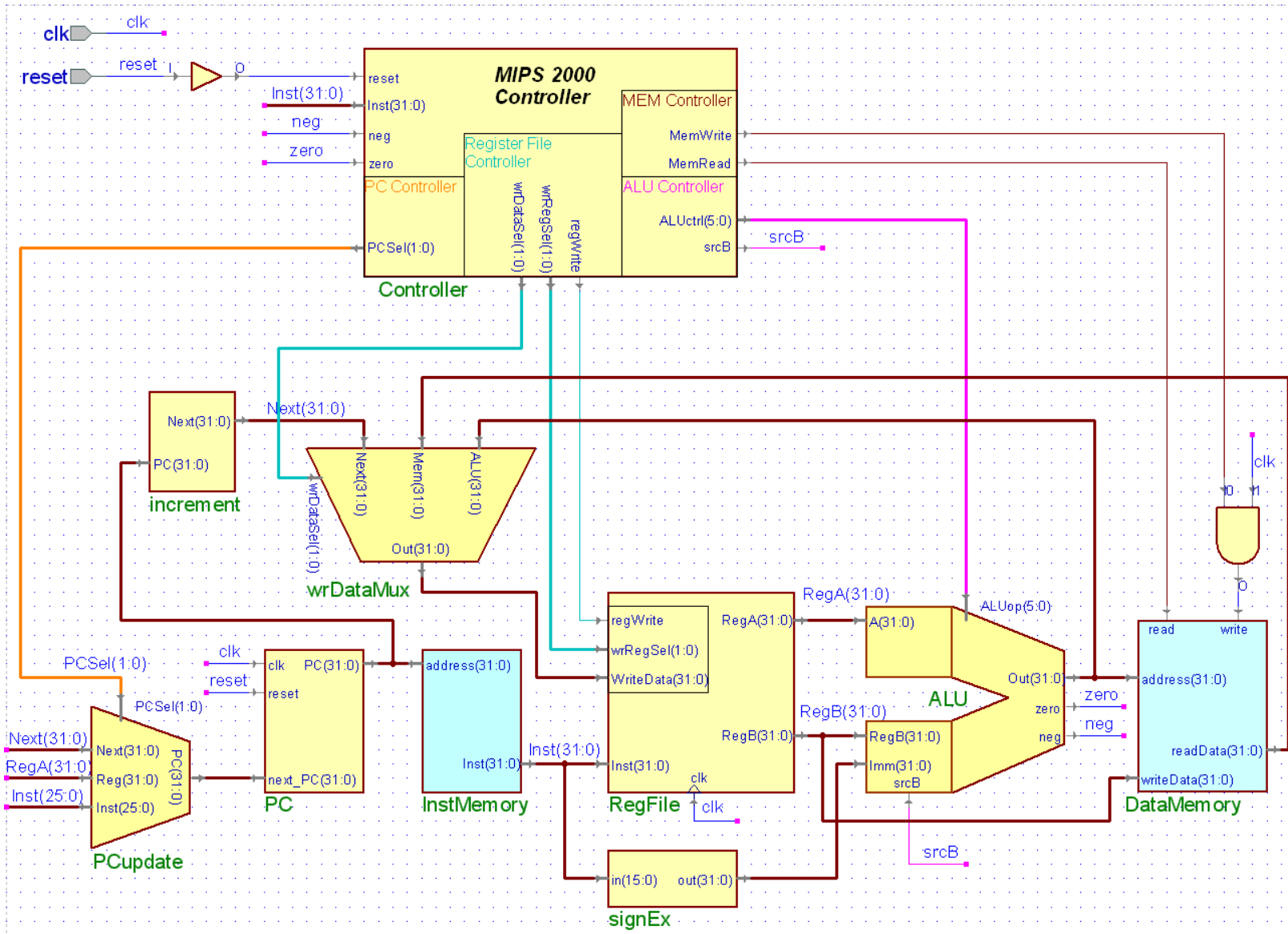


## SignExtender:

convert 16-bit to 32-bit



# Global View



# R-Format Operations

<b>Func</b>	<b>Operation</b>	<b>PC</b>	<b>comment</b>
ADD	$rd = rs + rt$	PC++	
SUB	$rd = rs - rt$	PC++	
SLT	$rd = (rs < rt) ? 1 : 0$	PC++	Set on less than
JR	No change	PC=rs	Jump to Register

## I-Format Ops:

	<b>Operation</b>	<b>PC</b>	<b>Comment</b>
ADDI	$rt = rs + SE(imm)$	PC++	
ORI	$rt = rs \mid imm$	PC++	
LUI	$rt = imm \ll 16$	PC++	Load upper immed
LW	$rt = MEM[rs + se(imm)]$	PC++	
SW	$MEM[rs + se(imm)] = rt$	PC++	
BEQ	$(rs == rt)?$	PC+1+(0 imm)	

## J-Format Ops:

<b>Func</b>	<b>Operation</b>	<b>PC</b>	<b>comment</b>
J		PC = target	a.k.a GOTO
JAL	r31 = PC+1	PC = target	Jump and Link

JAL stores next address, jumps to target (a.k.a fn call )

---

## Final Tips:

- Verilog uses “?” for Don't Cares
- Waveforms will make things easier
- Be sure to set clk and reset
- Simulation: 14000ns limit-- longer, controller is broken
- Don't use opcode defs from 378 text-- ours are different.



# Programming Example

Given:  $A$  is an array of size  $B$

Goal: Compute

$$\sum_{i=0}^B A[i]$$

Lets use a for loop ...

---

## In Assembly Language:

- Variables → Registers
- Array Access → Load (name+offset)
- Minimal Control Structures
  - Branches (  $A < B$ ,  $A \geq B$ ,  $A \neq B$  )
  - Jumps

# C to ASM

## High Level Language

```
C = 0;
for(i = 0; i < B; i = i+1) {
    C = C + A[i];
}
```

## Pseudo-Asm

```
C = 0;
i = 0;
Loop: bge i, B, Exit
    temp = A+i
    temp2 = load 0(temp)
    C = C + temp2;
    i = i + 1;
j Loop
Exit: ...
```

# ASM to RTL

```
C = 0;
  i = 0;
Loop: bge i, B, Exit
      temp = load A[i];
      C = C + temp;
      i = i + 1;
      j Loop
Exit: ...
```

```
r3 = r0, PC++
r4 = r0, PC++
Loop: PC =
      ( r4 ≥ r2) ? Exit : PC+1
r6 = MEM[r4+r1 ],PC++
r3 = r3 + r6, PC++
r4 = r4 + 1, PC++
PC = LOOP
Exit:
```