Computer Organization: A real processor

Steven Balensiefer

Background

- We built a model processor
  - 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

- R-format: | op | rs | rt | rd | shft | func |

  r3 = r1 + r2 | ALU | r1 | r2 | r3 | X | ADD|

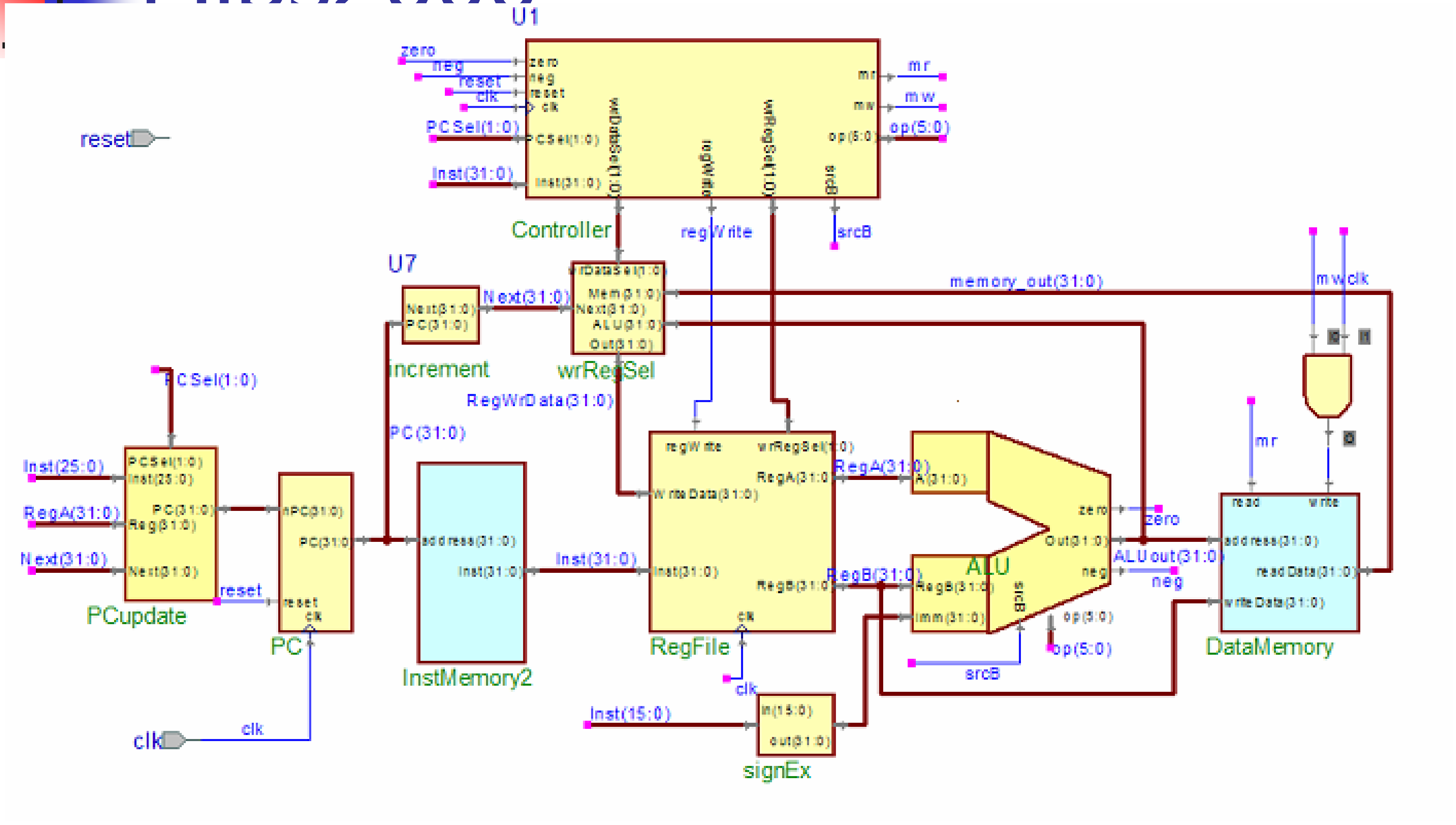
- I-format | op | rs | rt | addr/immediate|

  r3 = imm(r2) |Load| r2 | r3 | imm |

- J-format: |op | target address |

  jal hanoi | JAL | addr(hanoi) |

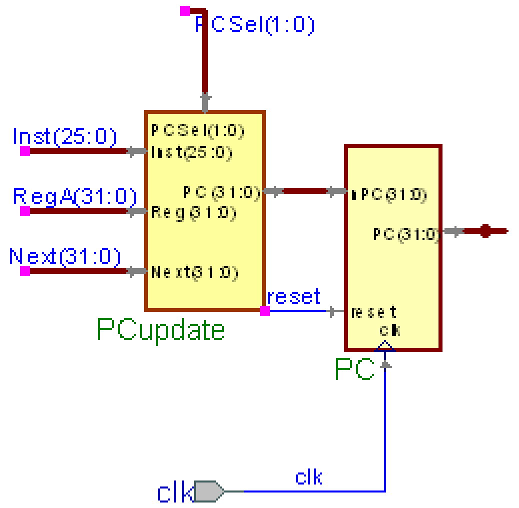
Mips2000



U1
Controller
zero
neg
reset
clk
PCSel(1:0)
Inst(31:0)
wrDataSel(1:0)
regWrite
wrRegSel(1:0)
srcB
mr
mw
op(5:0)
U7
increment
Next(31:0)
PC(31:0)
wrRegSel
Mem(31:0)
ALU Out(31:0)
memory_out(31:0)
mwclk
RegWrData(31:0)
PCupdate
Inst(25:0)
RegA(31:0)
PC
nPC(31:0)
InstMemory2
address(31:0)
RegFile
WriteData(31:0)
RegB(31:0)
ALU
A(31:0)
imm(31:0)
Out(31:0)
ALUout(31:0)
DataMemory
read
write
readData(31:0)
writeData(31:0)
signEx
Inst(15:0)
in(15:0)
out(31:0)

Program Counter

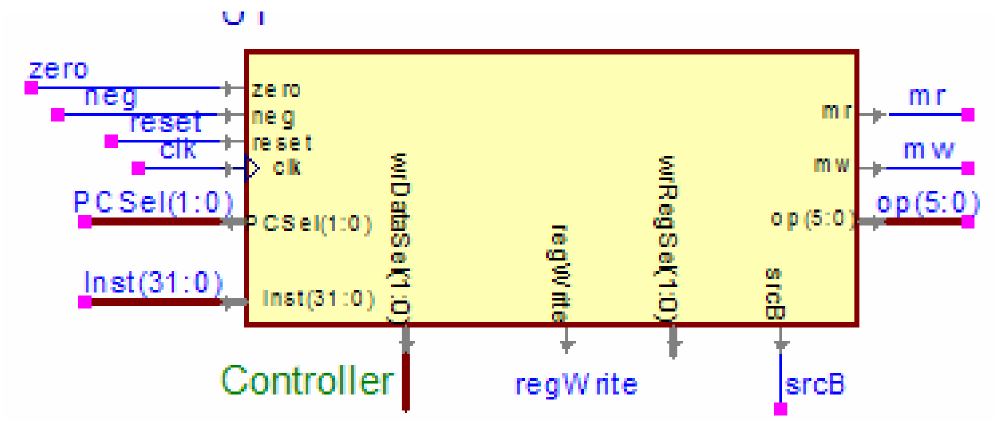
```
assign offset = {{16{Inst[15]}},Inst}; // sign
      extend the immediate
assign Branch = Next + offset;
assign Jump = {Next[31:26],Inst[25:0]}; //
      used by J instruction

  // 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

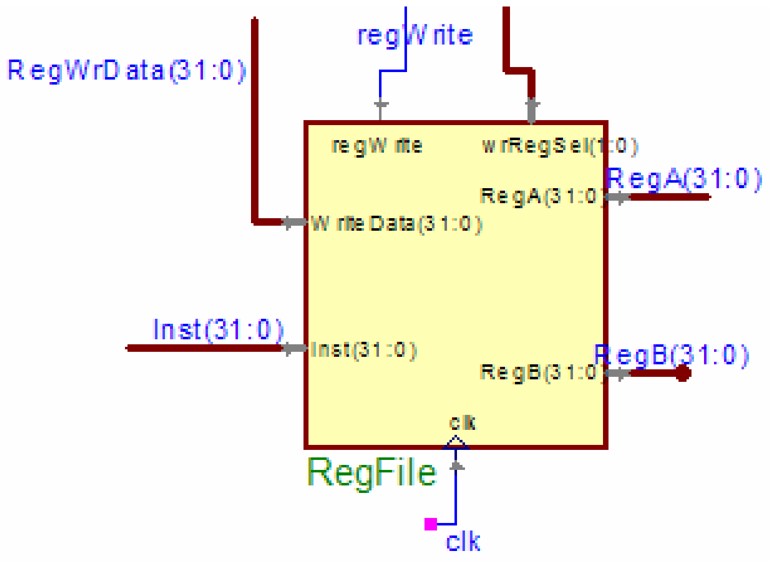
Skeleton Code:

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



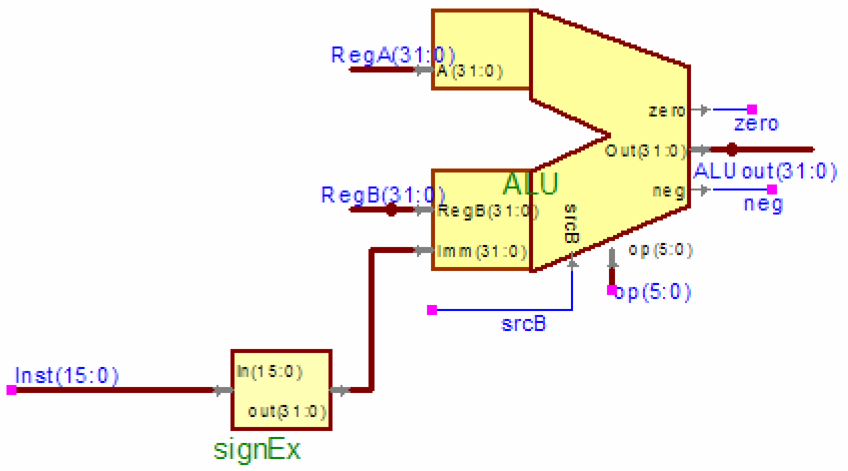
Register File

```
// 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 : Ir

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

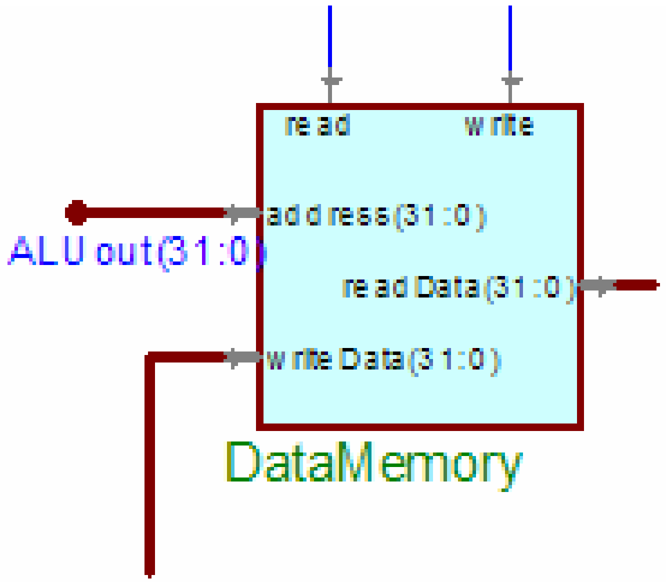


Data Memory

- Address from ALU
- Data from Reg B

- Memory-Mapped I/O
  - SW to xFFFFFFF
  - Buffers / Displays

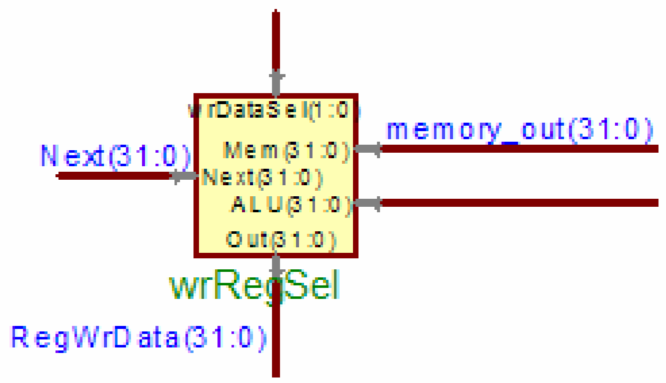
(See dmemory.v for more)



Miscellaneous

WrRegSel:

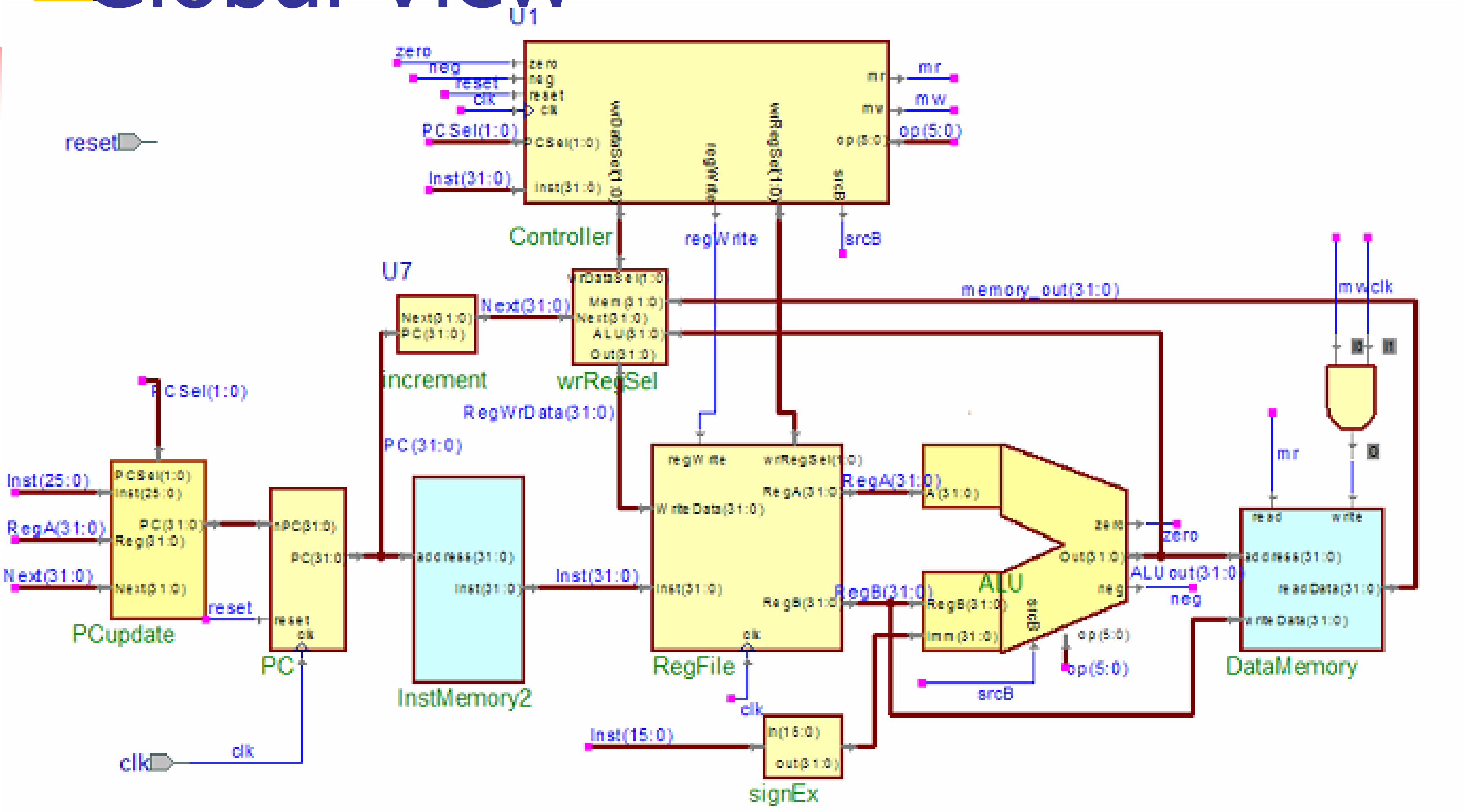
// wrDataSel
//   00: Out = ALU
//   01: Out = MEM
//   1X: Out = PC + 1



SignExtender:

convert 16-bit to 32-bit

Global View


U1
Controller
zero
neg
reset
clk
PCSel(1:0)
Inst(31:0)
mr
mw
op(5:0)
wrDataSel(1:0)
regWrite
wrRegSel(1:0)
srcB
U7
increment
Next(31:0)
PC(31:0)
wrRegSel
Mem(31:0)
ALU(31:0)
Out(31:0)
memory_out(31:0)
RegWrData(31:0)
PCupdate
Inst(25:0)
RegA(31:0)
Next(31:0)
PC
nPC(31:0)
InstMemory2
address(31:0)
RegFile
WriteData(31:0)
RegB(31:0)
ALU
Imm(31:0)
ALU out(31:0)
DataMemory
read
write
readData(31:0)
writeData(31:0)
signEx
Inst(15:0)
in(15:0)
out(31:0)
mwclk

R-Format Operations

Func	Operation	PC	comment
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:

	Operation	PC	Comment
ADDI	rt = rs+SE(imm)	PC++	
ORI	rt = rs | imm	PC++	
LUI	rt = imm << 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:

Func	Operation	PC	comment
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

Programming Example

Given: A is an array of size B

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

I'll just use a for loop …

Assembly Language

- Variables → Registers
- Array Access → Load (name+offset)
- Minimal Control Structures
  - Branches (A < B, A >= B, A != B)
  - Jumps

C to ASM

High Level Language

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

Psuedo-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:
```