THE UNIVERSITY OF KANSAS

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

EECS 645 – Computer Architecture

Fall 2016

Homework 10 (Single-Cycle MIPS)

Student Name: Student ID:

Homework 10

In this homework assignment you will be designing the Single-Cycle version of the MIPS processor that supports a subset, 10 instructions, of MIPS ISA by integrating the MIPS components that were covered through all previous homework assignments. The microarchitecture datapath and control path are shown in the following pages. The supported instructions of this version of MIPS are as follows:

a) 6 Arithmetic/Logical instructions: *add, sub, and, or, slt, addi*
b) 2 Memory reference: *lw, sw*
c) 2 Control transfer: *beq, j*

You are required to:

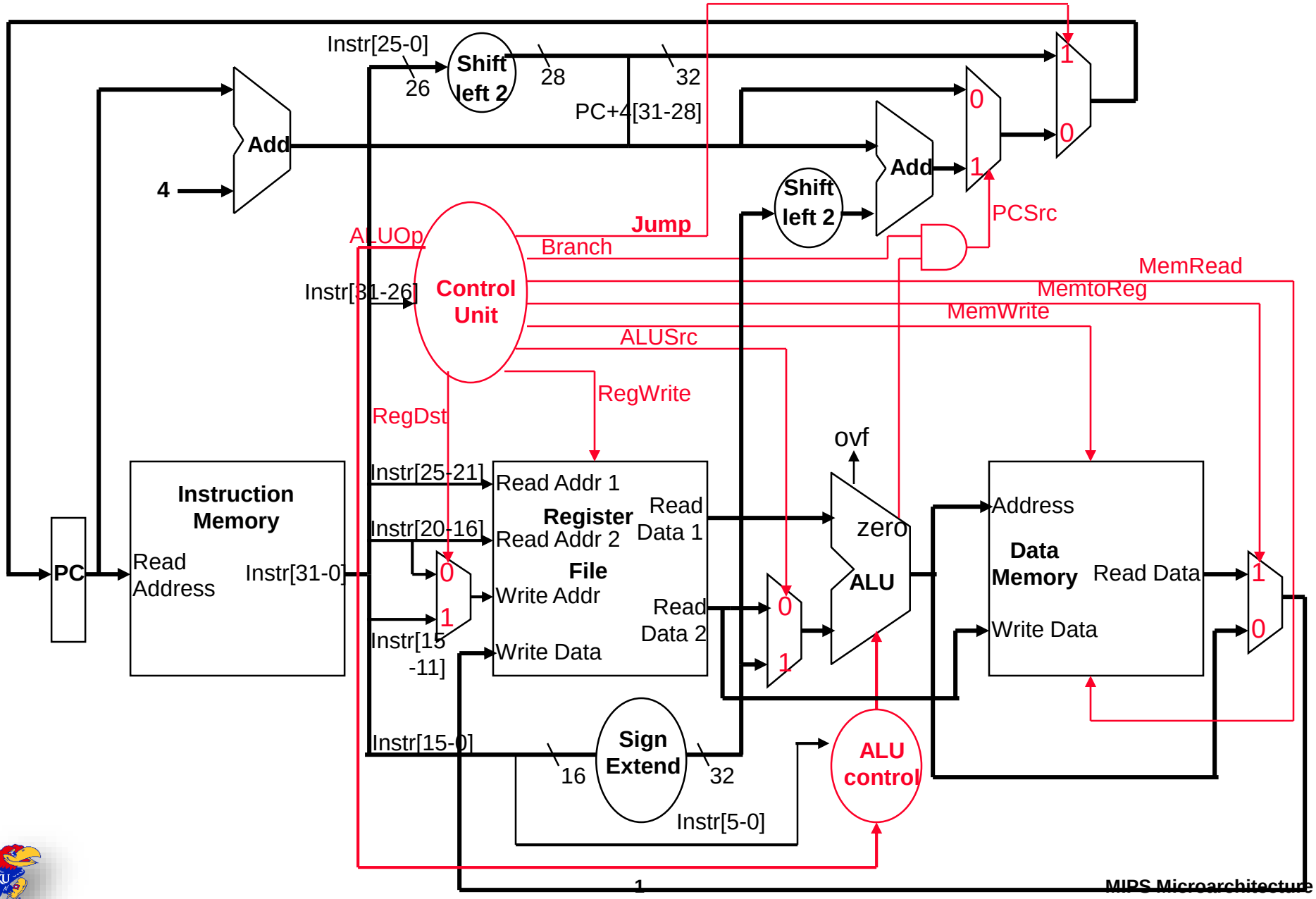
a) Write the equivalent **VHDL code**, and
b) Verify the correct operation through Vivado Simulator by comparing your simulation results with those of MARS runs.

Steps:

1) Download the file "HW10_MIPS_Single_Cycle.zip" from blackboard and extract its contents.
2) Launch Vivado and create a new project, for example "vivado_project", with the default settings.
3) Add to the project the VHDL design and simulation source folders; "\HW10_MIPS_Single_Cycle\06_MIPS_Single_Cycle\design_sources" and "\HW10_MIPS_Single_Cycle\06_MIPS_Single_Cycle\simulation_sources" respectively.
4) Edit the VHDL files in the folder "\HW10_MIPS_Single_Cycle\06_MIPS_Single_Cycle\design_sources\incomplete\" according to your design such that it describes the required MIPS microarchitecture.
5) Set the simulation time to the proper time, e.g. 300 ns, and then launch Vivado Simulator.
6) Verify the correctness of your design. You may go back to step 4 to correct your code until your design works properly as required.

Hint: You could use the provided assembly test program "\HW10_MIPS_Single_Cycle\06_MIPS_Single_Cycle\testing_options\program_assembly.asm" to verify your design by comparing your simulation results in Vivado with the run results of the same test program in MARS.

Complete Single-Cycle/Non-Pipelined Datapath



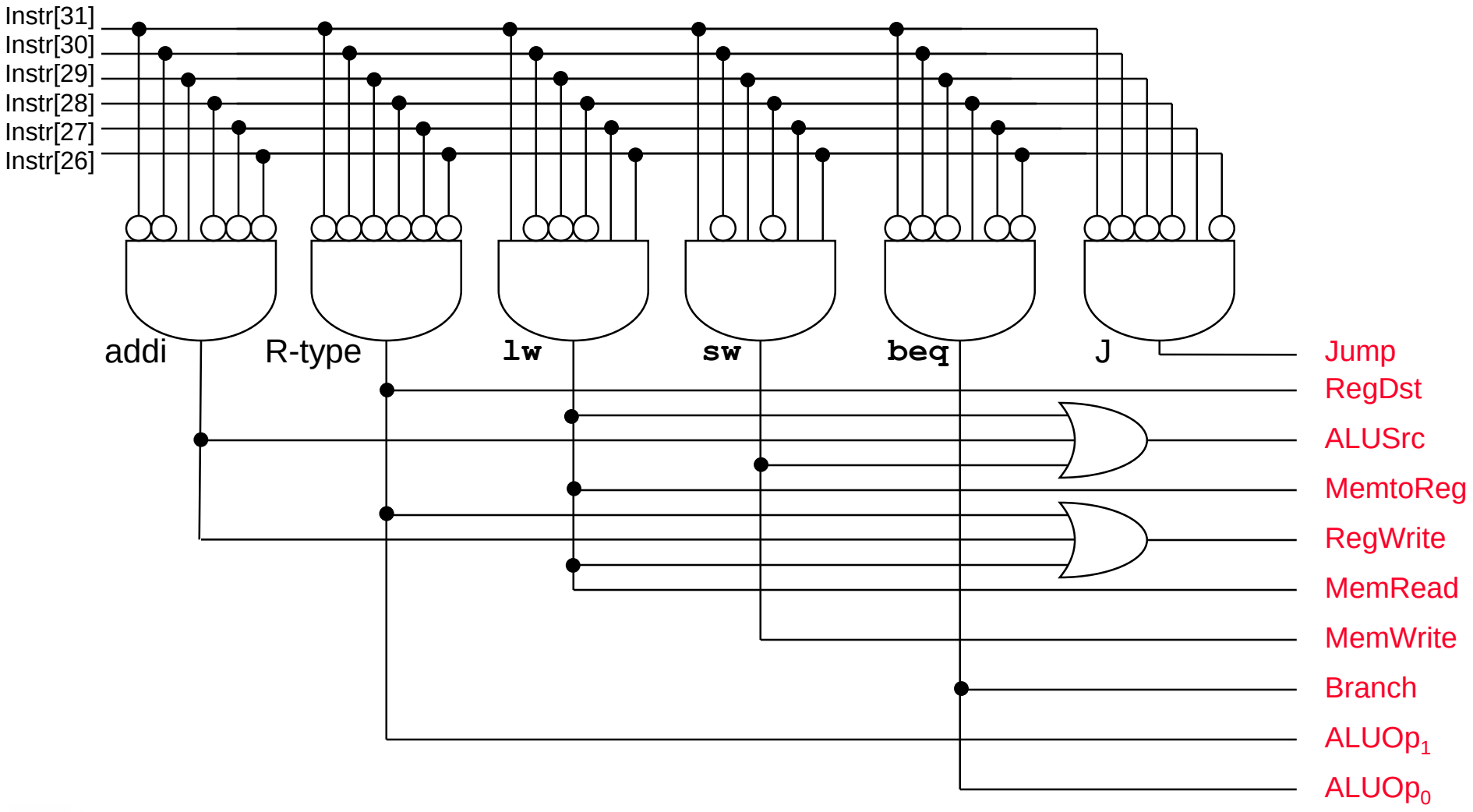
Main Control Unit

Instr. OP	RegDst	ALUSrc	MemToReg	RegWr	MemRd	MemWr	Branch	ALUOp	Jump
R-type 000000	1	0	0	1	0	0	0	10	0
lw 100011	0	1	1	1	1	0	0	00	0
sw 101011	X	1	X	0	0	1	0	00	0
beq 000100	X	0	X	0	0	0	1	01	0
j 000010	X	X	X	0	0	0	X	XX	1
addi 001000	0	1	0	1	0	0	0	00	0

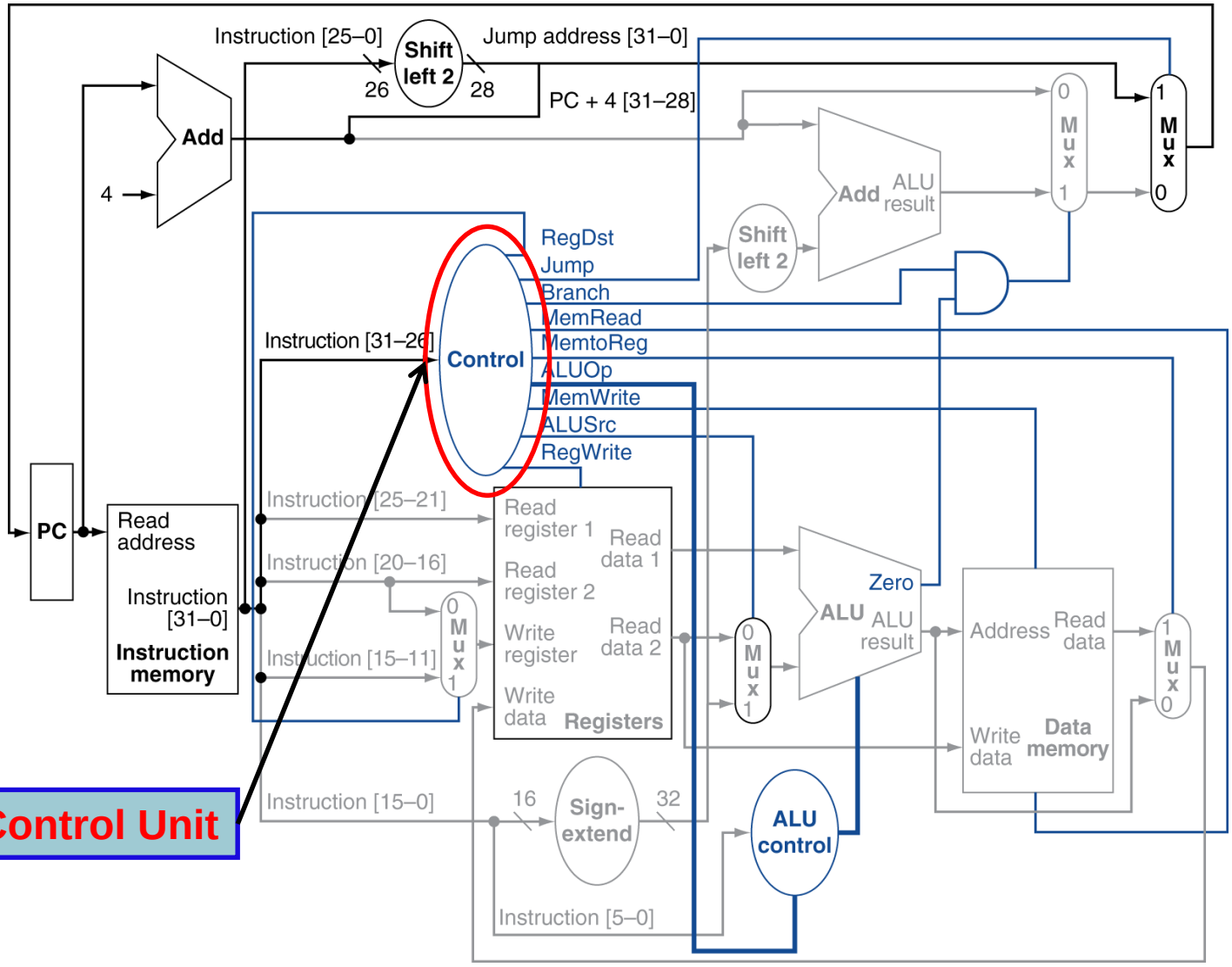
- Setting of the MemRd signal (for R-type, sw, beq, j) depends on the memory design (could have to be 0 or could be a X (don't care))

Main Control Unit

- From the truth table can design the Main Control logic

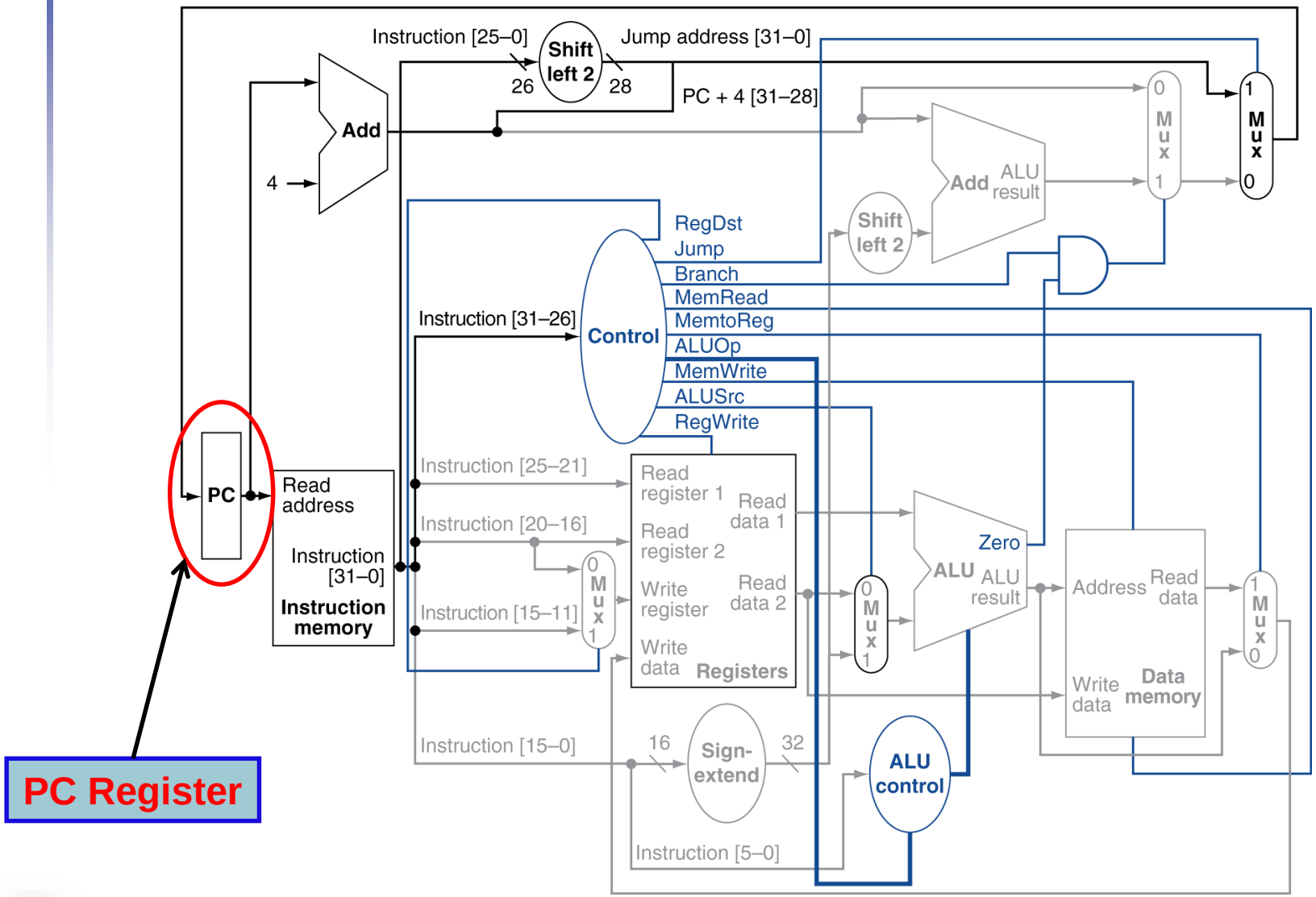


Single-Cycle/Non-Pipelined Datapath (Main Control Unit)



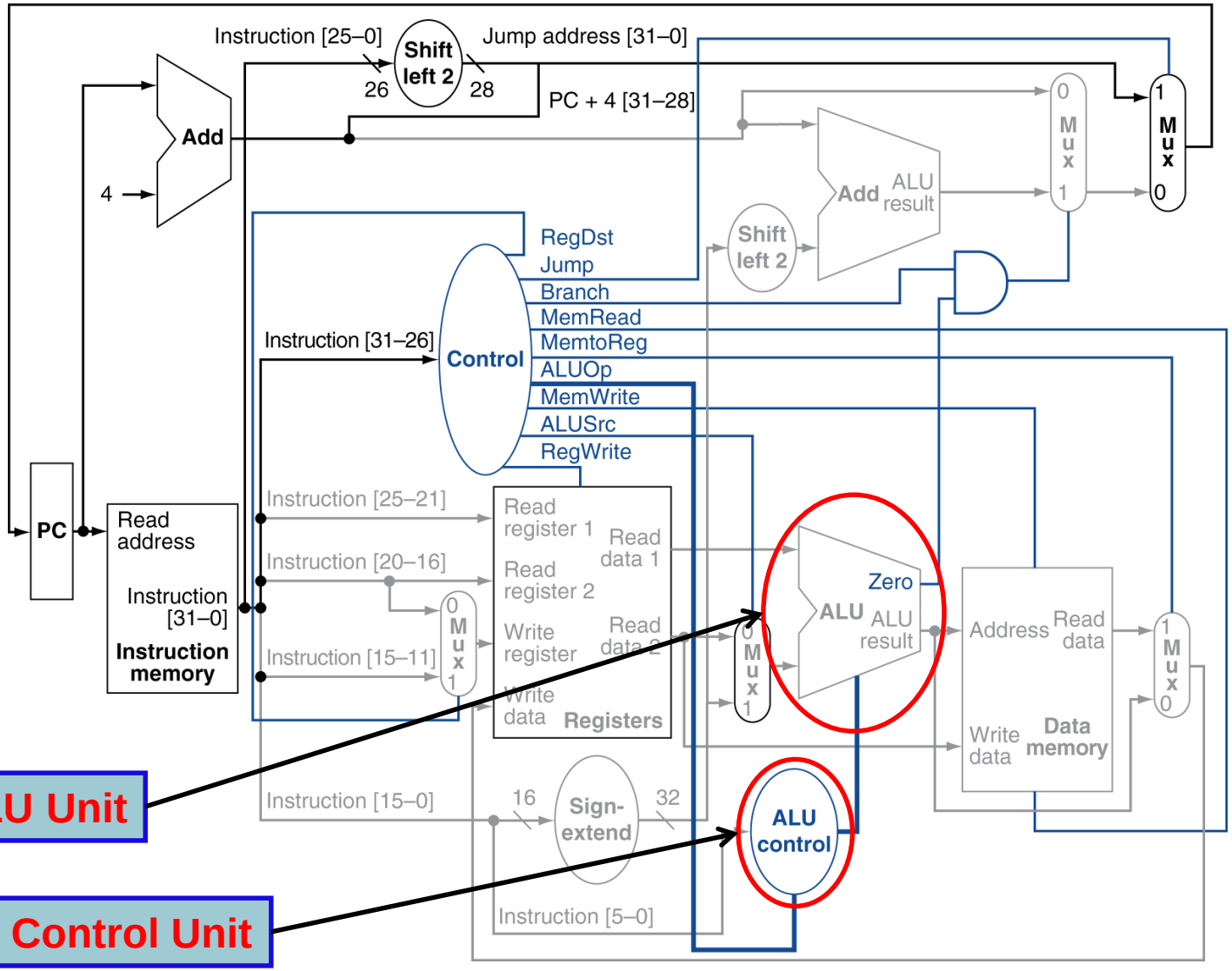
Main Control Unit

Single-Cycle/Non-Pipelined Datapath (PC Register)



PC Register

Single-Cycle/Non-Pipelined Datapath (ALU Unit & ALU Control Unit)



ALU Unit & ALU Control Unit

- Assume 2-bit ALUOp derived from opcode
  - Combinational logic derives ALU control

opcode	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0

opcode	ALUOp	Operation	funct	ALU function	ALU control
lw ≡ 100011	00	load word	XXXXXX	add	0010
sw ≡ 101011	00	store word	XXXXXX	add	0010
beq ≡ 000100	01	branch equal	XXXXXX	subtract	0110
R-type ≡ 000000	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

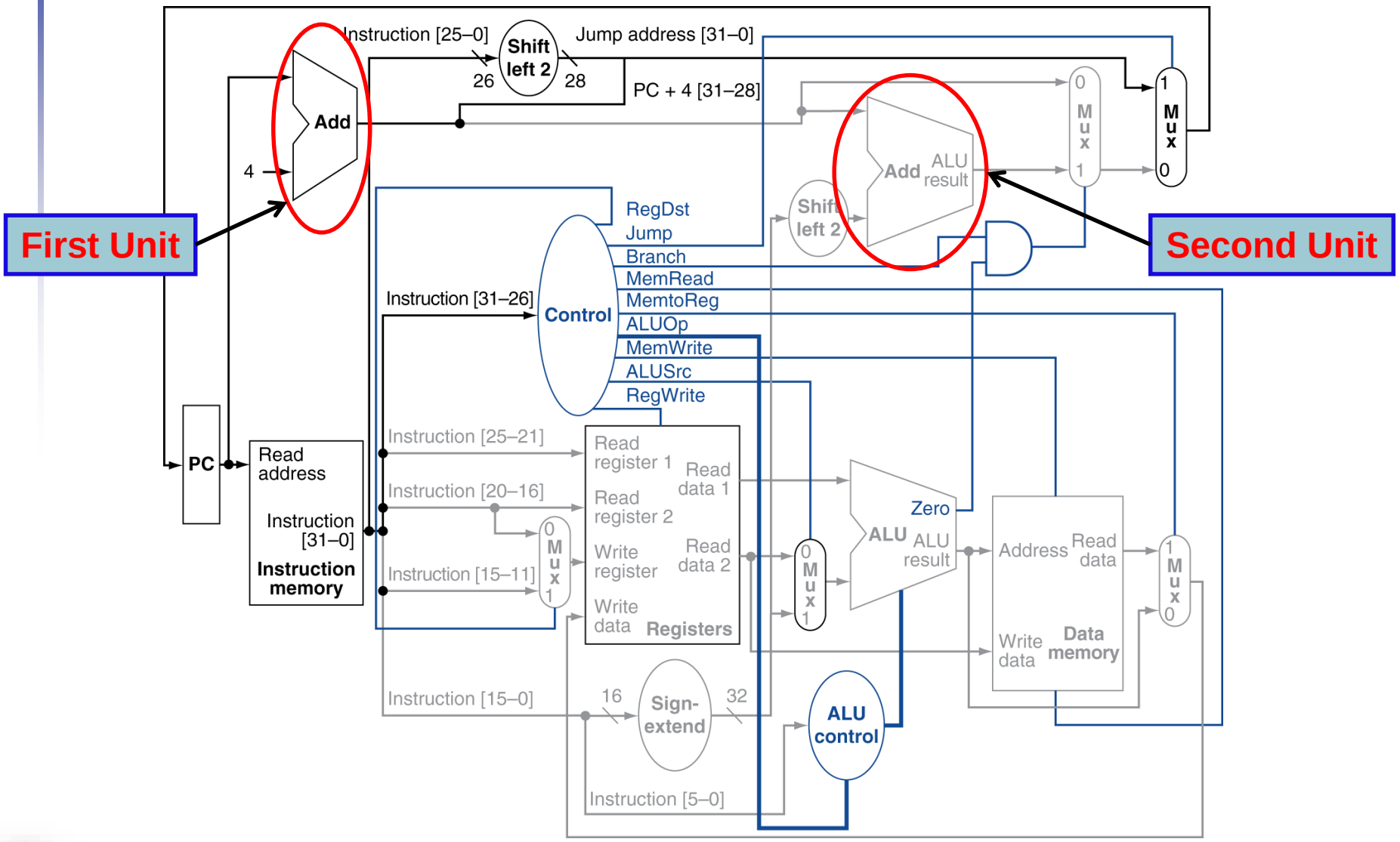
```
module MIPSALU (ALUctl, A, B, ALUOut, Zero);
   input [3:0] ALUctl;
   input [31:0] A,B;
   output reg [31:0] ALUOut;
   output Zero;
   assign Zero = (ALUOut==0); //Zero is true if ALUOut is 0
   always @(ALUctl, A, B) begin //reevaluate if these change
      case (ALUctl)
         0: ALUOut <= A & B;
         1: ALUOut <= A | B;
         2: ALUOut <= A + B;
         6: ALUOut <= A - B;
         7: ALUOut <= A < B ? 1 : 0;
         12: ALUOut <= ~(A | B); // result is nor
         default: ALUOut <= 0;
      endcase
    end
endmodule
```

FIGURE C.5.15 A Verilog behavioral definition of a MIPS ALU.

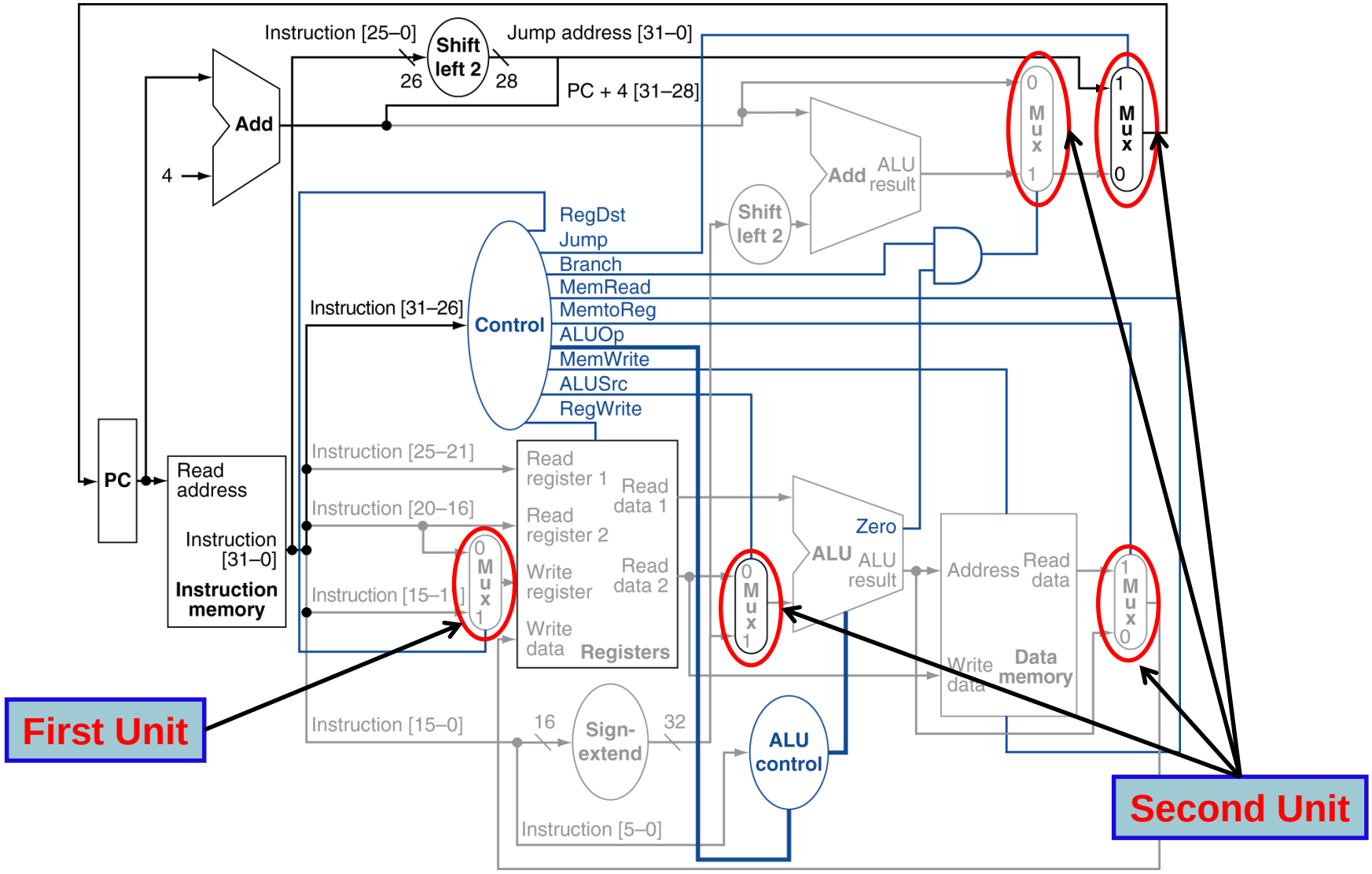
```
module ALUControl (ALUOp, FuncCode, ALUCtl);
   input [1:0] ALUOp;
   input [5:0] FuncCode;
   output [3:0] reg ALUCtl;
   always case (FuncCode)
      32: ALUCtl <=2; // add
      34: ALUCtl <=6; //subtract
      36: ALUCtl <=0; // and
      37: ALUCtl <=1; // or
      42: ALUCtl <=7; // slt
      default: ALUCtl <=15; // should not happen
  endcase
endmodule
```

FIGURE C.5.16 The MIPS ALU control: a simple piece of combinational control logic.

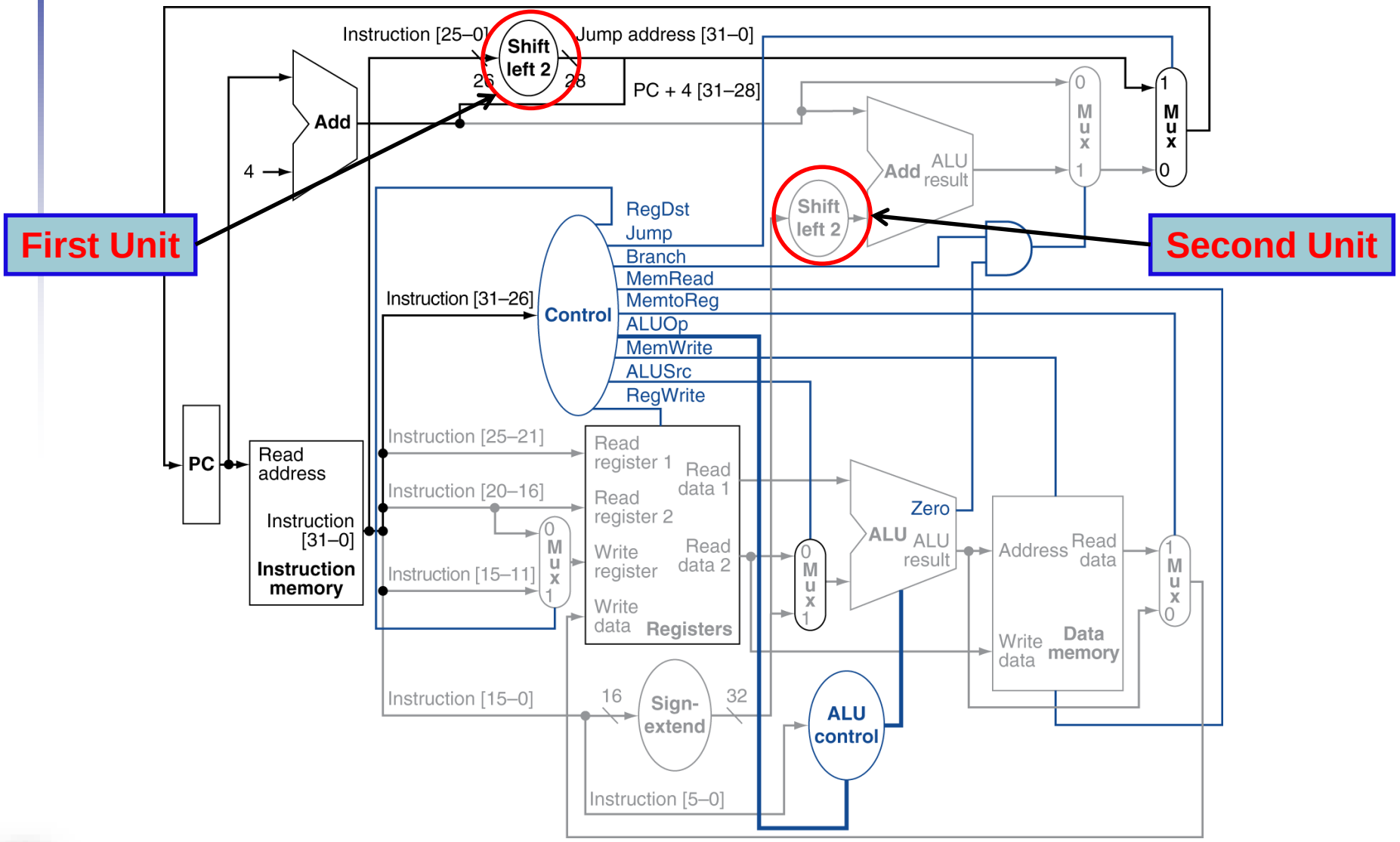
Single-Cycle/Non-Pipelined Datapath (Adder Units)



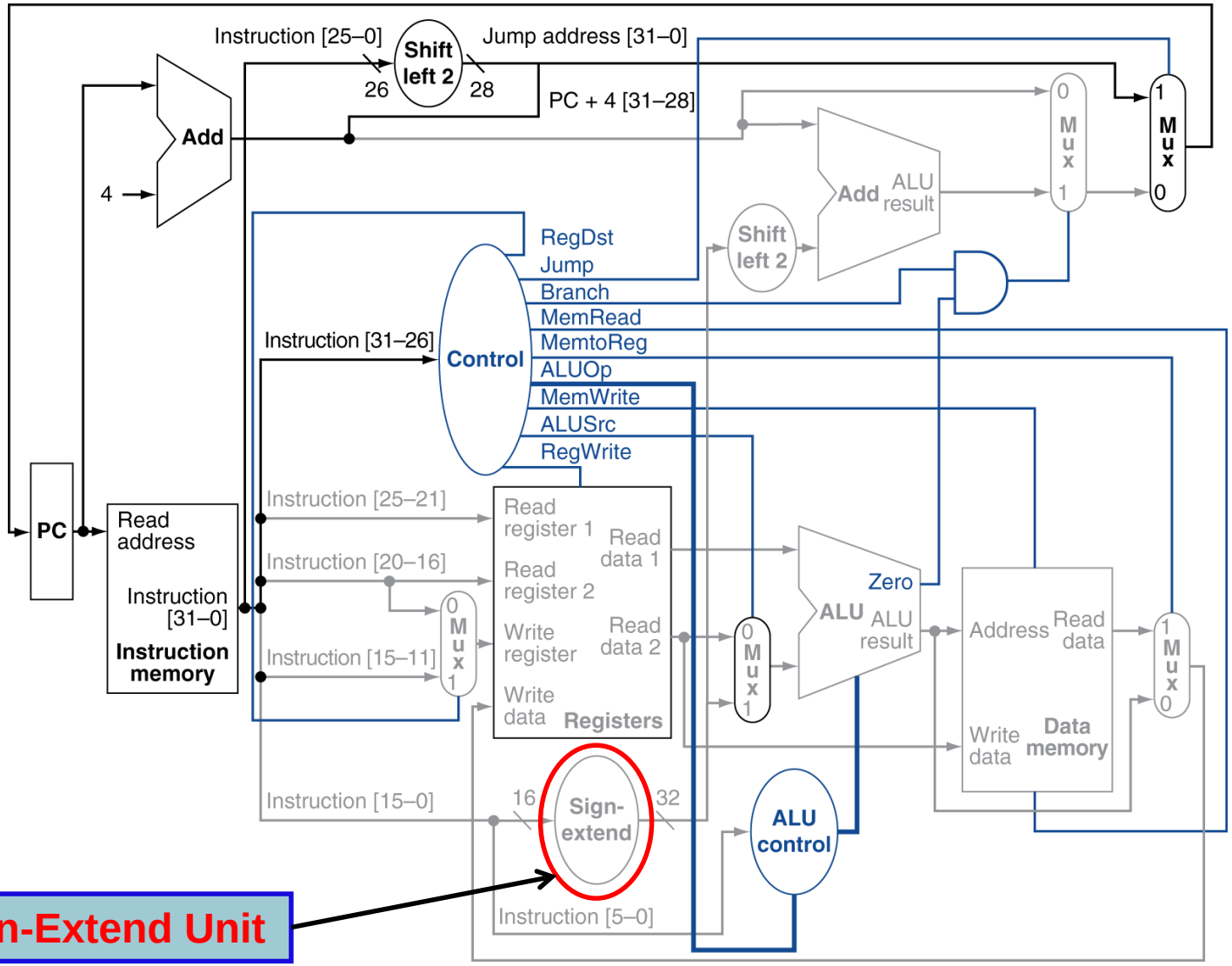
Single-Cycle/Non-Pipelined Datapath (Multiplexer Units)



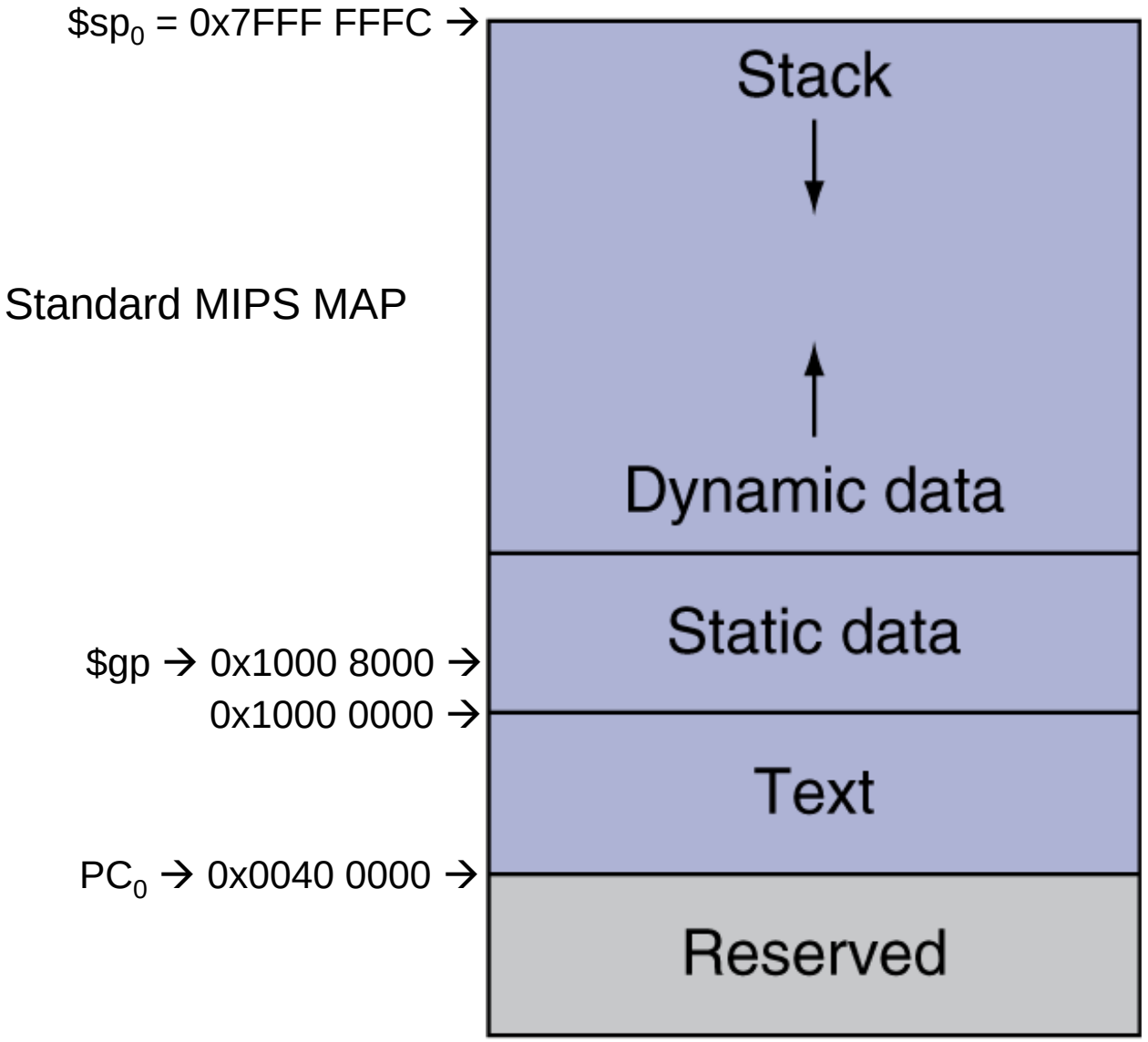
Single-Cycle/Non-Pipelined Datapath (Shift-Left Units)



Single-Cycle/Non-Pipelined Datapath (Sign-Extend Unit)



Review: MIPS Memory Layout/Map



Review: MIPS Memory Layout/Map

sp_0 = 0x7FFF FFFC →

sp_0 = 0x7FFF EFFC →

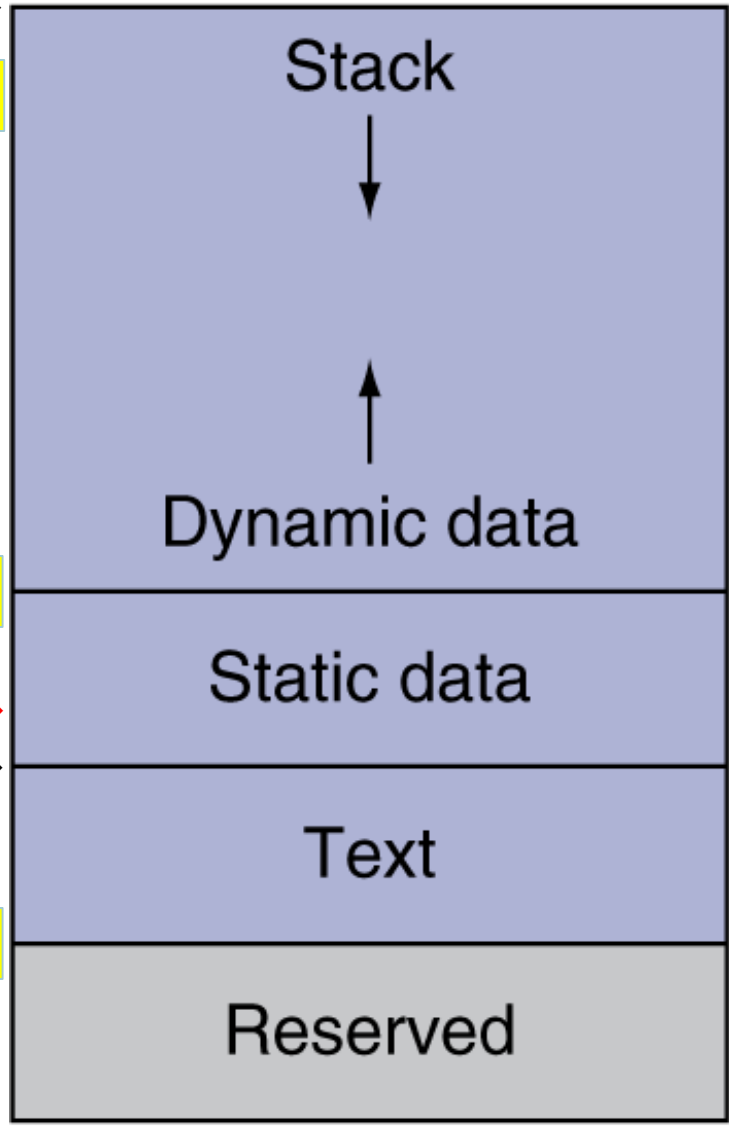
MIPS MAP in MARS

.data → 0x1001 0000 →

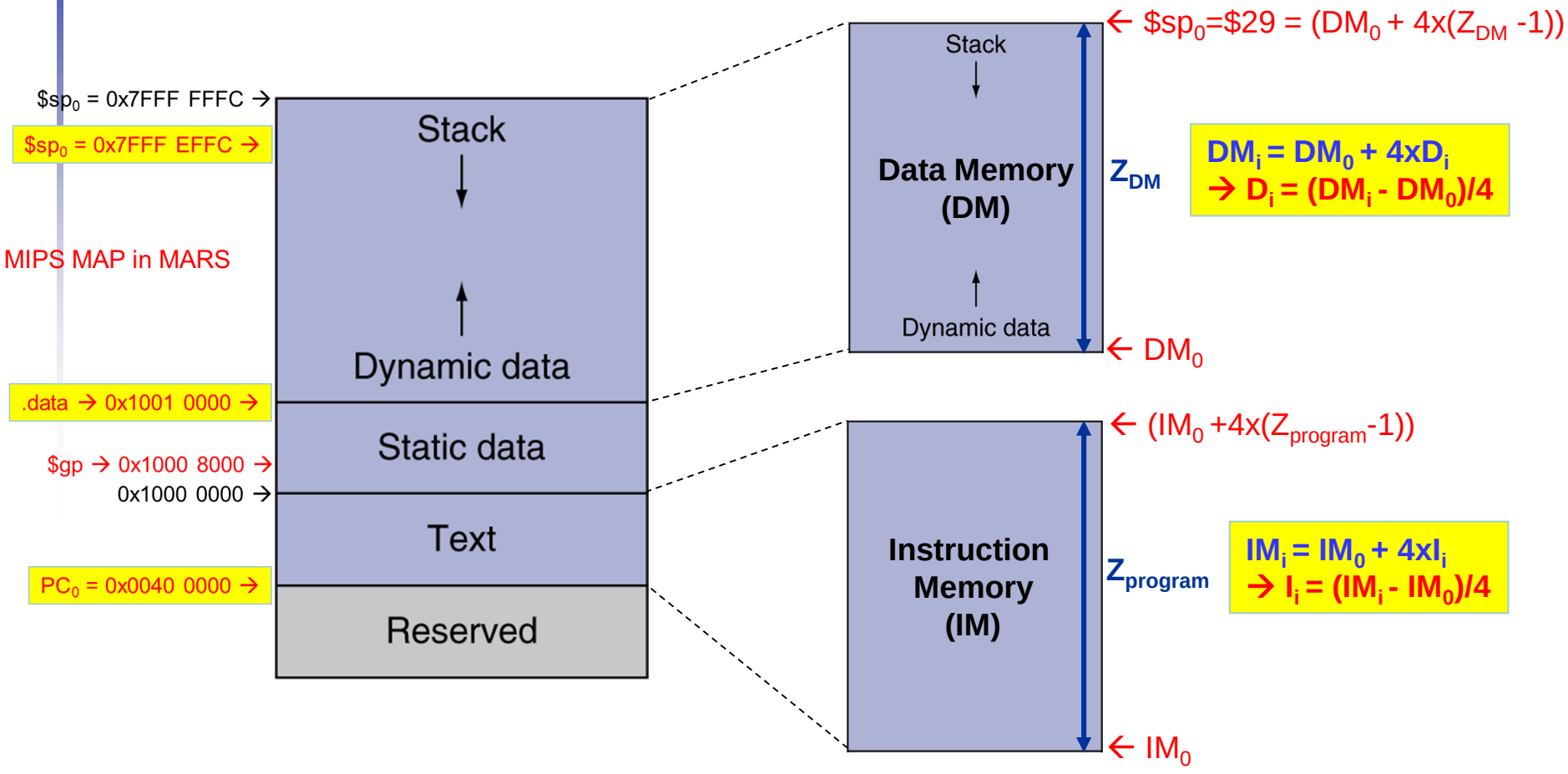
$gp → 0x1000 8000 →

0x1000 0000 →

PC_0 = 0x0040 0000 →



Review: Mapping Your Data & Program



Z_{DM} ≡ Allocated size of data memory in 32-bit words (locations)
DM_0 ≡ First address of data memory, could be anything, e.g. 0 or **0x1001 0000**

$Z_{program}$ ≡ Allocated size of instruction memory = Size of test program in 32-bit words (instructions)
IM_0 ≡ First address of instruction memory, could be anything, e.g. 0 or **0x0040 0000 = PC_0**

Review: Mapping Your Program

program_assembly.asm | program_assembly_option1.asm | program_assembly_option2.asm

```
1           addi    $t0, $zero, 5   # Instruction 00
2           addi    $t1, $zero, 7   # Instruction 01
3   start:  sw      $t0,  0($sp)    # Instruction 02
4           sw      $t1, -4($sp)    # Instruction 03
5           lw      $s0,  0($sp)    # Instruction 04
6           lw      $s1, -4($sp)    # Instruction 05
7           beq     $s0, $s1, Else  # Instruction 06
8           add     $s3, $s0, $s1   # Instruction 07
9           j       Exit            # Instruction 08
10  Else:   sub     $s3, $s0, $s1   # Instruction 09
11  Exit:   add     $s0, $s0, $s3   # Instruction 10
12          or      $s1, $s1, $s3   # Instruction 11
13          addi    $t0, $t0,  3    # Instruction 12
14          addi    $t1, $t1,  3    # Instruction 13
15          addi    $sp, $sp, -8    # Instruction 14
16          j       start           # Instruction 15
```

Review: Mapping Your Program - Option 1

program_assembly.asm | program_assembly_option1.asm | program_assembly_option2.asm

```
1          addi    $t0, $zero, 5    # Instruction 00 --> Address 00 = x"00000000"
2          addi    $t1, $zero, 7    # Instruction 01 --> Address 04 = x"00000004"
3   start: sw      $t0,  0($sp)     # Instruction 02 --> Address 08 = x"00000008"
4          sw      $t1, -4($sp)     # Instruction 03 --> Address 12 = x"0000000C"
5          lw      $s0,  0($sp)     # Instruction 04 --> Address 16 = x"00000010"
6          lw      $s1, -4($sp)     # Instruction 05 --> Address 20 = x"00000014"
7          beq     $s0, $s1, Else   # Instruction 06 --> Address 24 = x"00000018"
8          add     $s3, $s0, $s1    # Instruction 07 --> Address 28 = x"0000001C"
9          j       Exit             # Instruction 08 --> Address 32 = x"00000020"
10  Else:  sub     $s3, $s0, $s1    # Instruction 09 --> Address 36 = x"00000024"
11  Exit:  add     $s0, $s0, $s3    # Instruction 10 --> Address 40 = x"00000028"
12         or      $s1, $s1, $s3    # Instruction 11 --> Address 44 = x"0000002C"
13         addi    $t0, $t0,  3     # Instruction 12 --> Address 48 = x"00000030"
14         addi    $t1, $t1,  3     # Instruction 13 --> Address 52 = x"00000034"
15         addi    $sp, $sp, -8     # Instruction 14 --> Address 56 = x"00000038"
16         j       start            # Instruction 15 --> Address 60 = x"0000003C"
```

$IM_i = IM_0 + 4xI_i$
→ $I_i = (IM_i - IM_0)/4$

$IM_0 = 0$, $IM_i = PC_i$
→ $I_i = PC_i /4$

$DM_i = DM_0 + 4xD_i$
→ $D_i = (DM_i - DM_0)/4$

$DM_0 = A_0 = 0$, $DM_i = A_i$
→ $I_i = A_i /4$

$\$sp_0 = DM_0 + 4x(Z_{DM}-1)$

$DM_0 = A_0 = 0$
→ $\$sp_0 = 4x(Z_{DM}-1)$

Review: Mapping Your Program - Option 1

```
 1  00100000000001000000000000000101
 2  00100000000001001000000000000111
 3  10101111101010000000000000000000
 4  10101111101010011111111111111100
 5  10001111101100000000000000000000
 6  10001111101100011111111111111100
 7  00010010000100010000000000000010
 8  00000010000100011001100000100000
 9  000010  00000000000000000000001010
10  00000010000100011001100000100010
11  00000010000100111000000000100000
12  00000010001100111000100000100101
13  00100001000010000000000000000011
14  00100001001010010000000000000011
15  00100011101111011111111111111000
16  000010  00000000000000000000000010
```

Manually-Assembled Program

```
 1  00100000000001000000000000000101
 2  00100000000001001000000000000111
 3  10101111101010000000000000000000
 4  10101111101010011111111111111100
 5  10001111101100000000000000000000
 6  10001111101100011111111111111100
 7  00010010000100010000000000000010
 8  00000010000100011001100000100000
 9  000010  00000010000000000000001010
10  00000010000100011001100000100010
11  00000010000100111000000000100000
12  00000010001100111000100000100101
13  00100001000010000000000000000011
14  00100001001010010000000000000011
15  00100011101111011111111111111000
16  000010  00000010000000000000000010
```

MARS-Assembled Program

Different

$IM_i = IM_0 + 4xI_i$
→ $I_i = (IM_i - IM_0)/4$

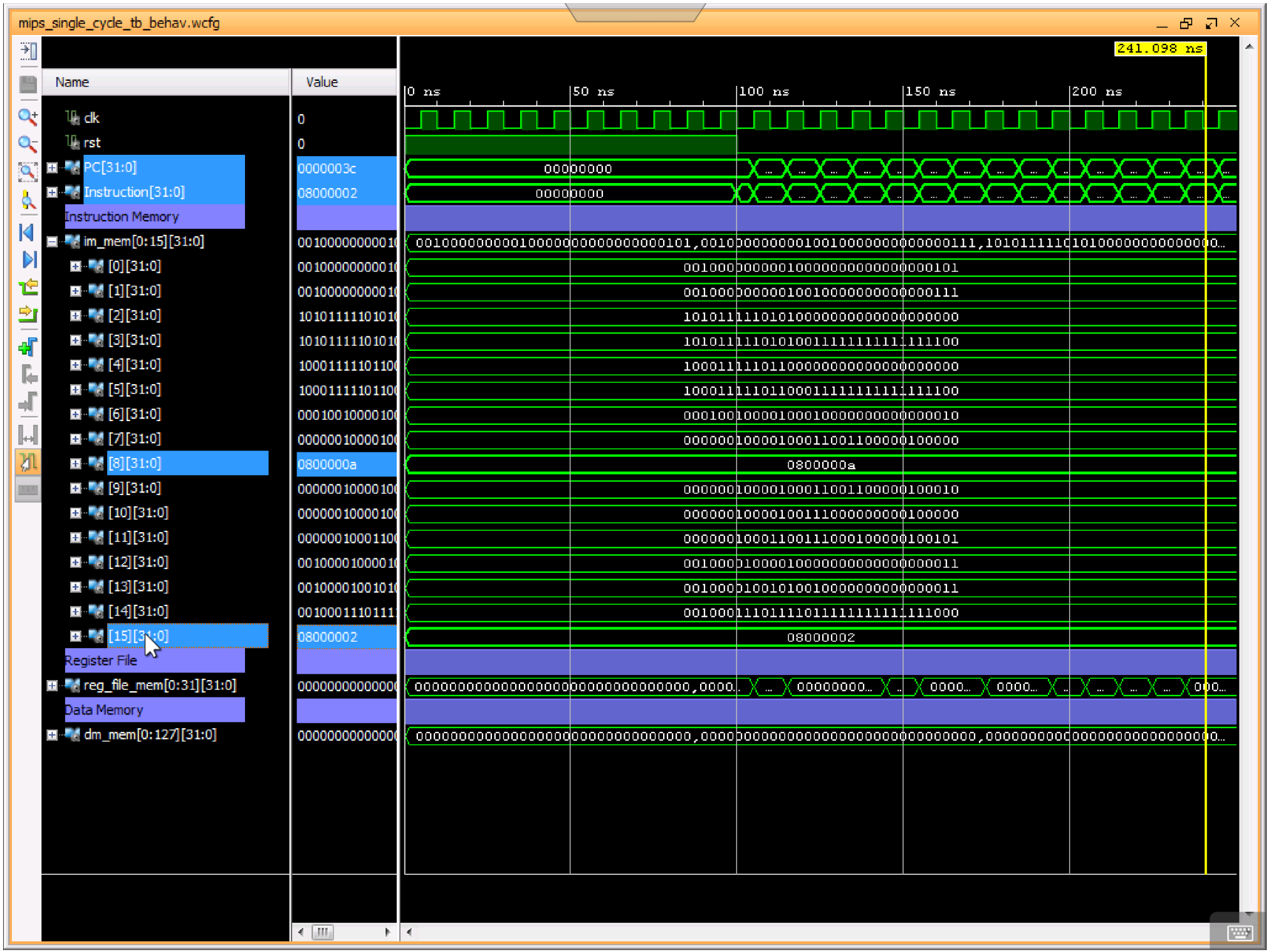
$IM_0 = 0$, $IM_i = PC_i$
→ $I_i = PC_i /4$

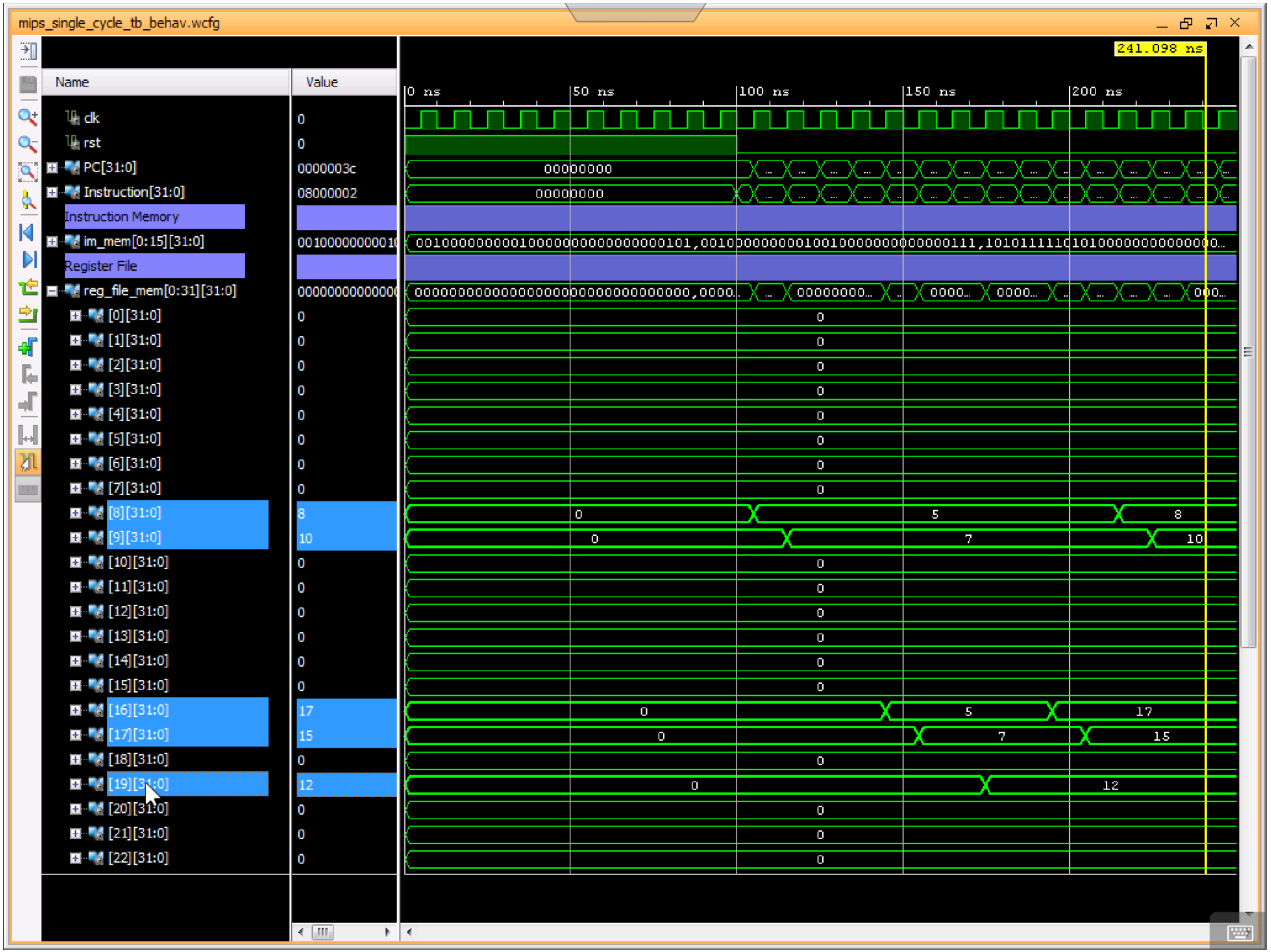
$DM_i = DM_0 + 4xD_i$
→ $D_i = (DM_i - DM_0)/4$

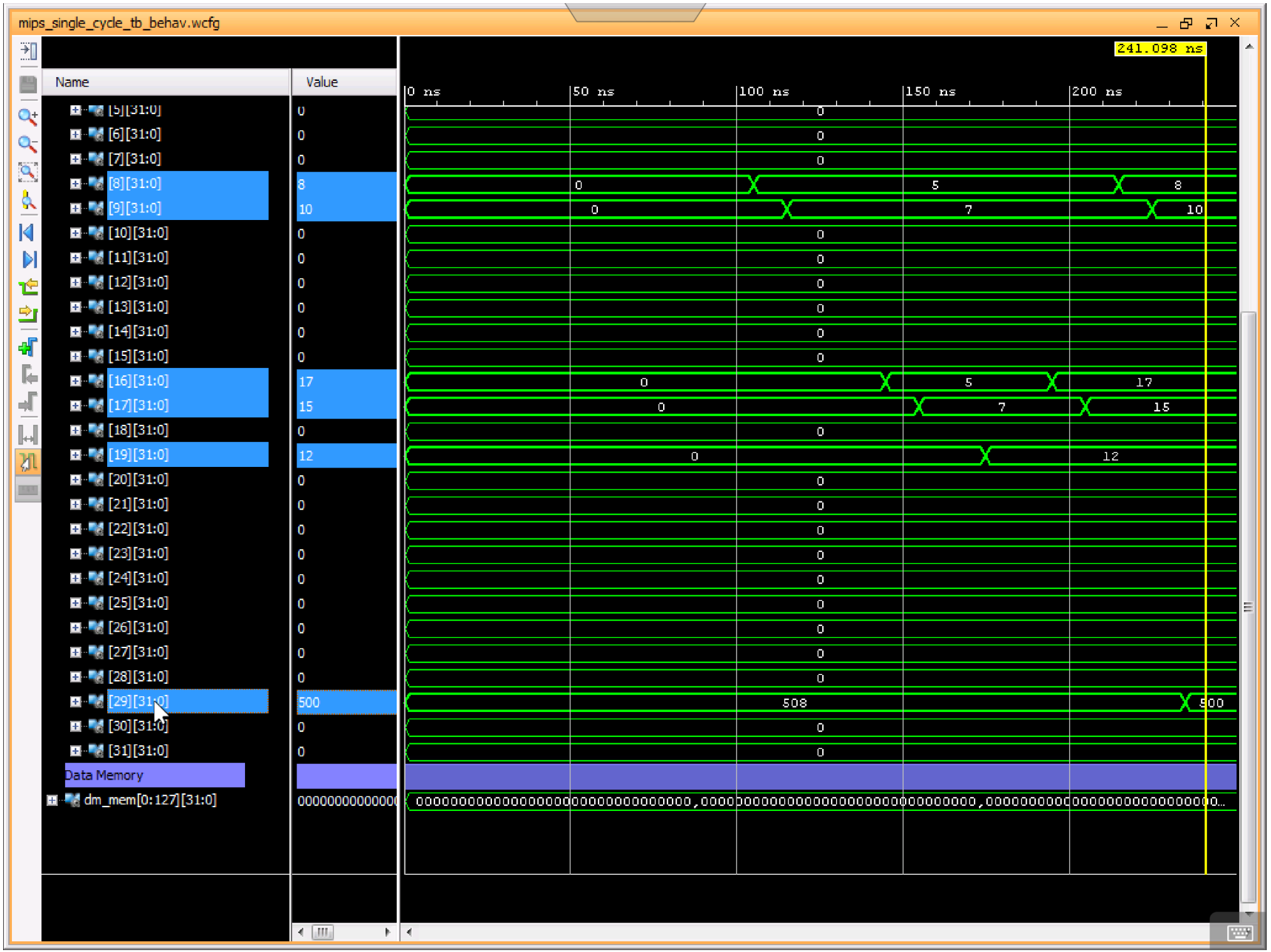
$DM_0 = A_0 = 0$, $DM_i = A_i$
→ $I_i = A_i /4$

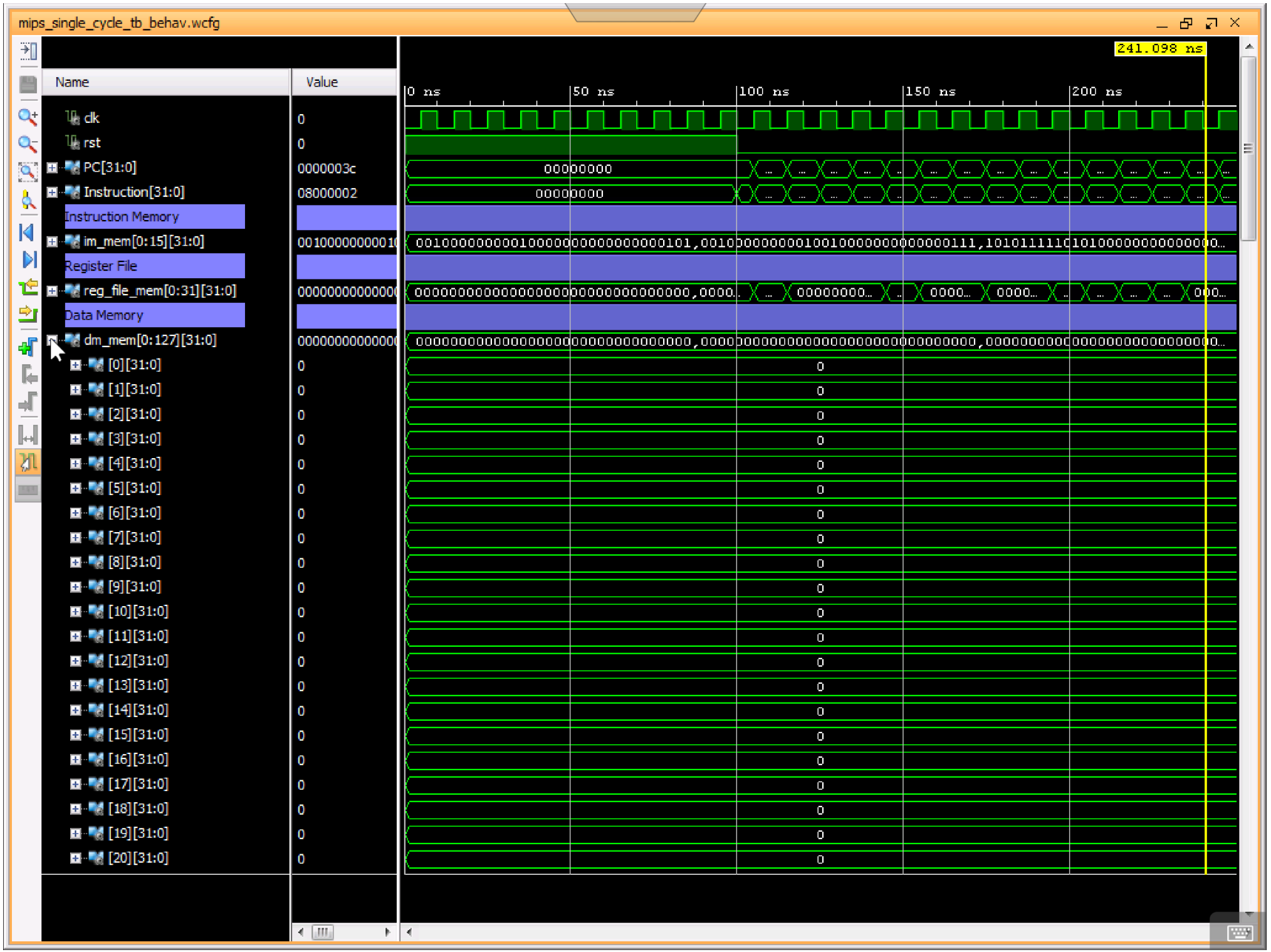
$\$sp_0 = DM_0 + 4x(Z_{DM}-1)$

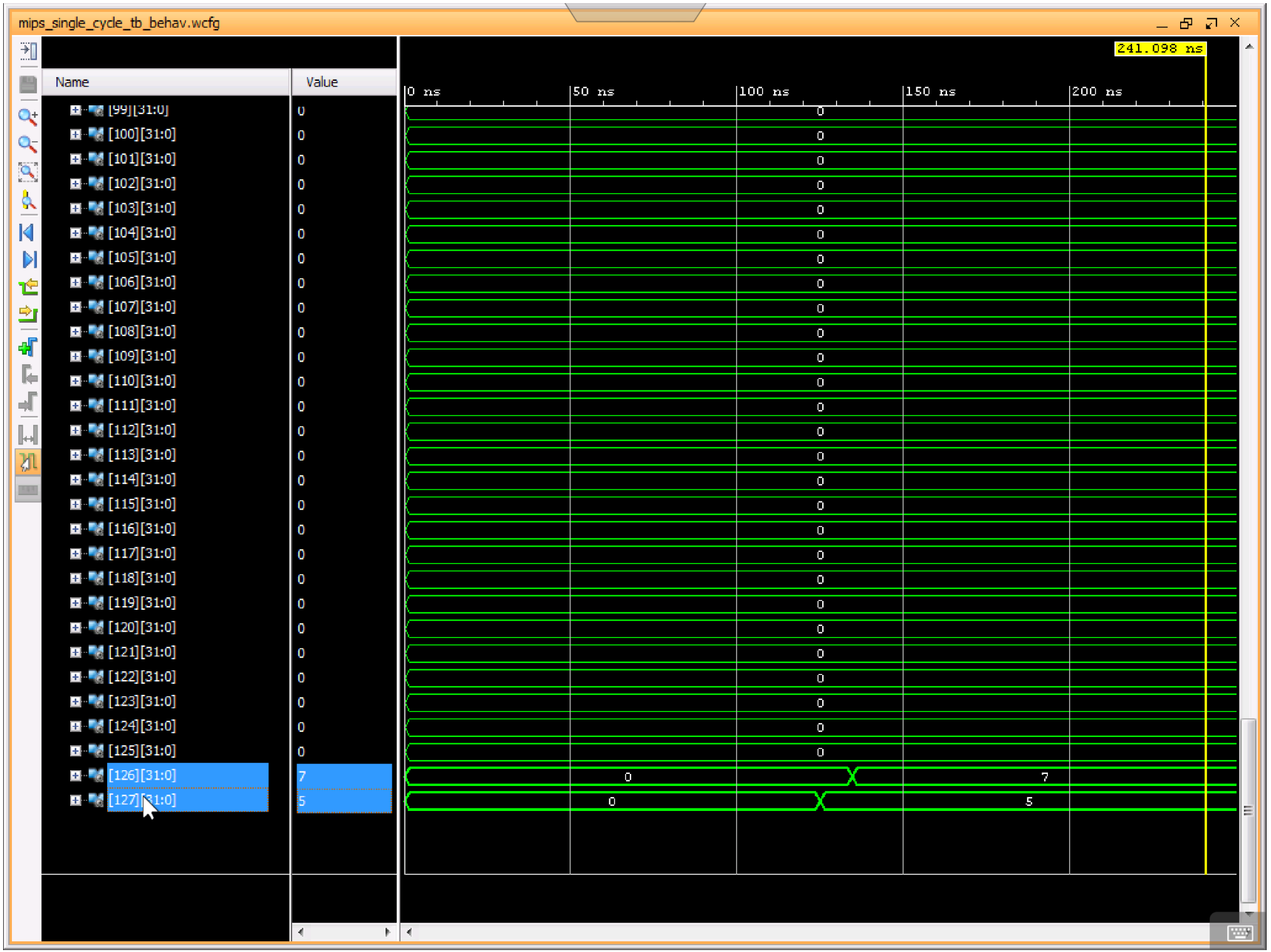
$DM_0 = A_0 = 0$
→ $\$sp_0 = 4x(Z_{DM}-1)$


mips_single_cycle_tb_behav.wcfg
Name
Value
241.098 ns
0 ns
50 ns
100 ns
150 ns
200 ns
clk
rst
PC[31:0]
Instruction[31:0]
Instruction Memory
im_mem[0:15][31:0]
[0][31:0]
[1][31:0]
[2][31:0]
[3][31:0]
[4][31:0]
[5][31:0]
[6][31:0]
[7][31:0]
[8][31:0]
[9][31:0]
[10][31:0]
[11][31:0]
[12][31:0]
[13][31:0]
[14][31:0]
[15][31:0]
Register File
reg_file_mem[0:31][31:0]
Data Memory
dm_mem[0:127][31:0]
0
0
0000003c
08000002
00000000
00000000
0800000a
08000002


mips_single_cycle_tb_behav.wcfg
Name
Value
241.098 ns
0 ns
50 ns
100 ns
150 ns
200 ns
clk
rst
PC[31:0]
Instruction[31:0]
Instruction Memory
im_mem[0:15][31:0]
Register File
reg_file_mem[0:31][31:0]
[0][31:0]
[1][31:0]
[2][31:0]
[3][31:0]
[4][31:0]
[5][31:0]
[6][31:0]
[7][31:0]
[8][31:0]
[9][31:0]
[10][31:0]
[11][31:0]
[12][31:0]
[13][31:0]
[14][31:0]
[15][31:0]
[16][31:0]
[17][31:0]
[18][31:0]
[19][31:0]
[20][31:0]
[21][31:0]
[22][31:0]
0000003c
08000002
00000000
00000000
8
10
17
15
12


mips_single_cycle_tb_behav.wcfg
Name
Value
241.098 ns
0 ns
50 ns
100 ns
150 ns
200 ns
[5][31:0]
[6][31:0]
[7][31:0]
[8][31:0]
[9][31:0]
[10][31:0]
[11][31:0]
[12][31:0]
[13][31:0]
[14][31:0]
[15][31:0]
[16][31:0]
[17][31:0]
[18][31:0]
[19][31:0]
[20][31:0]
[21][31:0]
[22][31:0]
[23][31:0]
[24][31:0]
[25][31:0]
[26][31:0]
[27][31:0]
[28][31:0]
[29][31:0]
[30][31:0]
[31][31:0]
Data Memory
dm_mem[0:127][31:0]
500
508


mips_single_cycle_tb_behav.wcfg
Name
Value
241.098 ns
0 ns
50 ns
100 ns
150 ns
200 ns
clk
rst
PC[31:0]
Instruction[31:0]
Instruction Memory
im_mem[0:15][31:0]
Register File
reg_file_mem[0:31][31:0]
Data Memory
dm_mem[0:127][31:0]
[0][31:0]
[1][31:0]
[2][31:0]
[3][31:0]
[4][31:0]
[5][31:0]
[6][31:0]
[7][31:0]
[8][31:0]
[9][31:0]
[10][31:0]
[11][31:0]
[12][31:0]
[13][31:0]
[14][31:0]
[15][31:0]
[16][31:0]
[17][31:0]
[18][31:0]
[19][31:0]
[20][31:0]


mips_single_cycle_tb_behav.wcfg
Name
Value
241.098 ns
0 ns
50 ns
100 ns
150 ns
200 ns
[100][31:0]
[101][31:0]
[102][31:0]
[103][31:0]
[104][31:0]
[105][31:0]
[106][31:0]
[107][31:0]
[108][31:0]
[109][31:0]
[110][31:0]
[111][31:0]
[112][31:0]
[113][31:0]
[114][31:0]
[115][31:0]
[116][31:0]
[117][31:0]
[118][31:0]
[119][31:0]
[120][31:0]
[121][31:0]
[122][31:0]
[123][31:0]
[124][31:0]
[125][31:0]
[126][31:0]
[127][31:0]
7
5

Review: Mapping Your Program - Option 2

program_assembly.asm | program_assembly_option1.asm | program_assembly_option2.asm

```
1          addi    $t0, $zero, 5   # Instruction 00 --> Address (00 + x"00400000") = x"00400000"
2          addi    $t1, $zero, 7   # Instruction 01 --> Address (04 + x"00400000") = x"00400004"
3  start:  sw      $t0,  0($sp)    # Instruction 02 --> Address (08 + x"00400000") = x"00400008"
4          sw      $t1, -4($sp)    # Instruction 03 --> Address (12 + x"00400000") = x"0040000C"
5          lw      $s0,  0($sp)    # Instruction 04 --> Address (16 + x"00400000") = x"00400010"
6          lw      $s1, -4($sp)    # Instruction 05 --> Address (20 + x"00400000") = x"00400014"
7          beq     $s0, $s1, Else  # Instruction 06 --> Address (24 + x"00400000") = x"00400018"
8          add     $s3, $s0, $s1   # Instruction 07 --> Address (28 + x"00400000") = x"0040001C"
9          j       Exit            # Instruction 08 --> Address (32 + x"00400000") = x"00400020"
10 Else:   sub     $s3, $s0, $s1   # Instruction 09 --> Address (36 + x"00400000") = x"00400024"
11 Exit:   add     $s0, $s0, $s3   # Instruction 10 --> Address (40 + x"00400000") = x"00400028"
12         or      $s1, $s1, $s3   # Instruction 11 --> Address (44 + x"00400000") = x"0040002C"
13         addi    $t0, $t0,  3    # Instruction 12 --> Address (48 + x"00400000") = x"00400030"
14         addi    $t1, $t1,  3    # Instruction 13 --> Address (52 + x"00400000") = x"00400034"
15         addi    $sp, $sp, -8    # Instruction 14 --> Address (56 + x"00400000") = x"00400038"
16         j       start           # Instruction 15 --> Address (60 + x"00400000") = x"0040003C"
```

$IM_i = IM_0 + 4xI_i$
→ $I_i = (IM_i - IM_0)/4$

$IM_0 = PC_0$ = 0x0040 0000 , $IM_i = PC_i$
→ $I_i = (PC_i - PC_0)/4$

$DM_i = DM_0 + 4xD_i$
→ $D_i = (DM_i - DM_0)/4$

$DM_0 = A_0$ = 0x1001 0000 , $DM_i = A_i$
→ $D_i = (A_i - A_0)/4$

$\$sp_0 = DM_0 + 4x(Z_{DM}-1)$

$DM_0 = A_0$ = 0x1001 0000
→ $\$sp_0 = A_0 + 4x(Z_{DM}-1)$

Review: Mapping Your Program - Option 2

```
 1  00100000000001000000000000000101
 2  00100000000001001000000000000111
 3  10101111101010000000000000000000
 4  10101111101010011111111111111100
 5  10001111101100000000000000000000
 6  10001111101100011111111111111100
 7  00010010000100010000000000000010
 8  00000010000100011001100000100000
 9  000010  00000100000000000000001010
10  00000010000100011001100000100010
11  00000010000100111000000000100000
12  00000010001100111000100000100101
13  00100001000010000000000000000011
14  00100001001010010000000000000011
15  00100011101111011111111111111000
16  000010  00000100000000000000000010
```

Manually-Assembled Program

```
 1  00100000000001000000000000000101
 2  00100000000001001000000000000111
 3  10101111101010000000000000000000
 4  10101111101010011111111111111100
 5  10001111101100000000000000000000
 6  10001111101100011111111111111100
 7  00010010000100010000000000000010
 8  00000010000100011001100000100000
 9  000010  00000100000000000000001010
10  00000010000100011001100000100010
11  00000010000100111000000000100000
12  00000010001100111000100000100101
13  00100001000010000000000000000011
14  00100001001010010000000000000011
15  00100011101111011111111111111000
16  000010  00000100000000000000000010
```

MARS-Assembled Program

Same

$IM_i = IM_0 + 4xI_i$
→ $I_i = (IM_i - IM_0)/4$

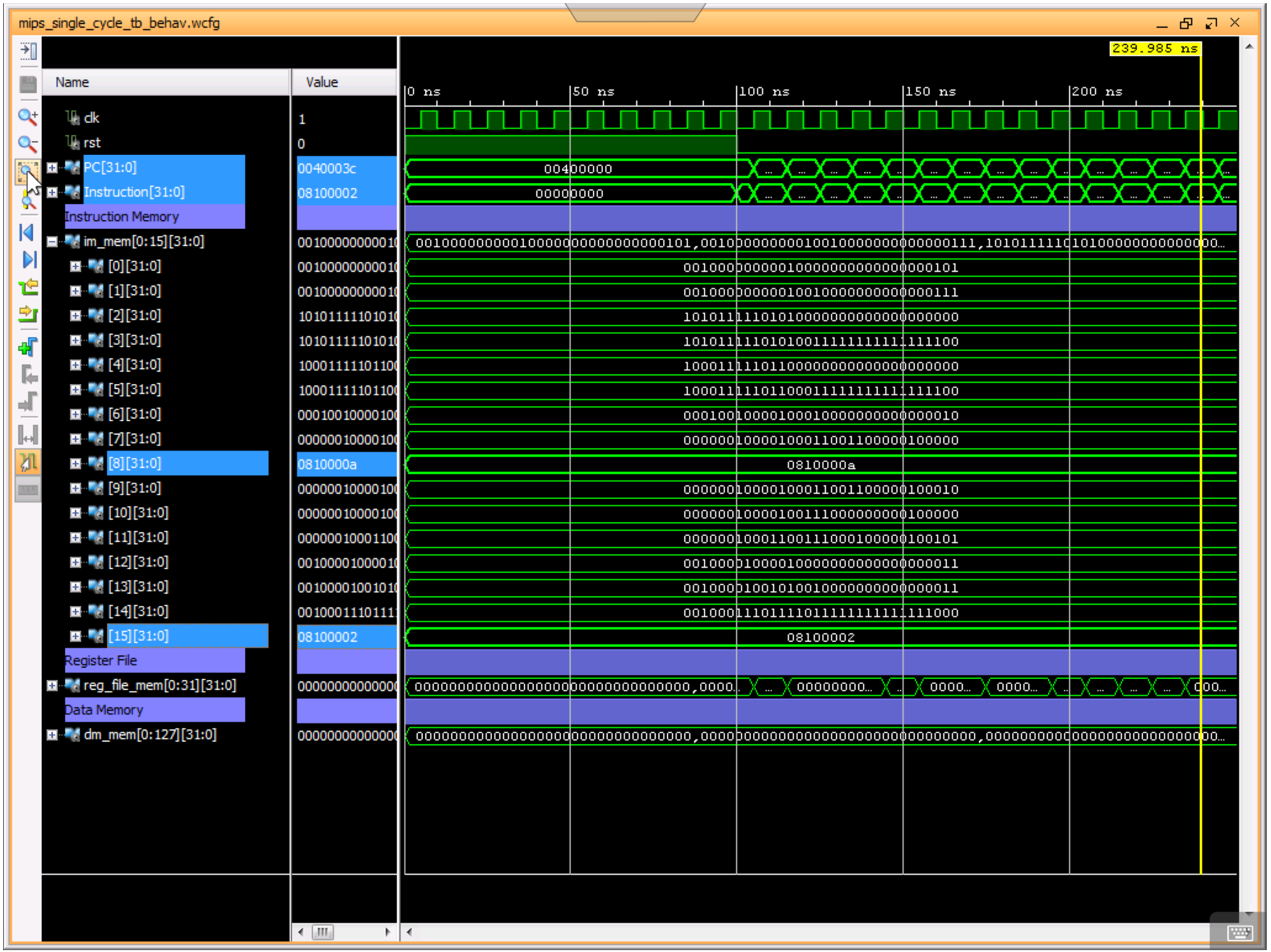
$IM_0 = PC_0 = 0x0040\ 0000$, $IM_i = PC_i$
→ $I_i = (PC_i - PC_0)/4$

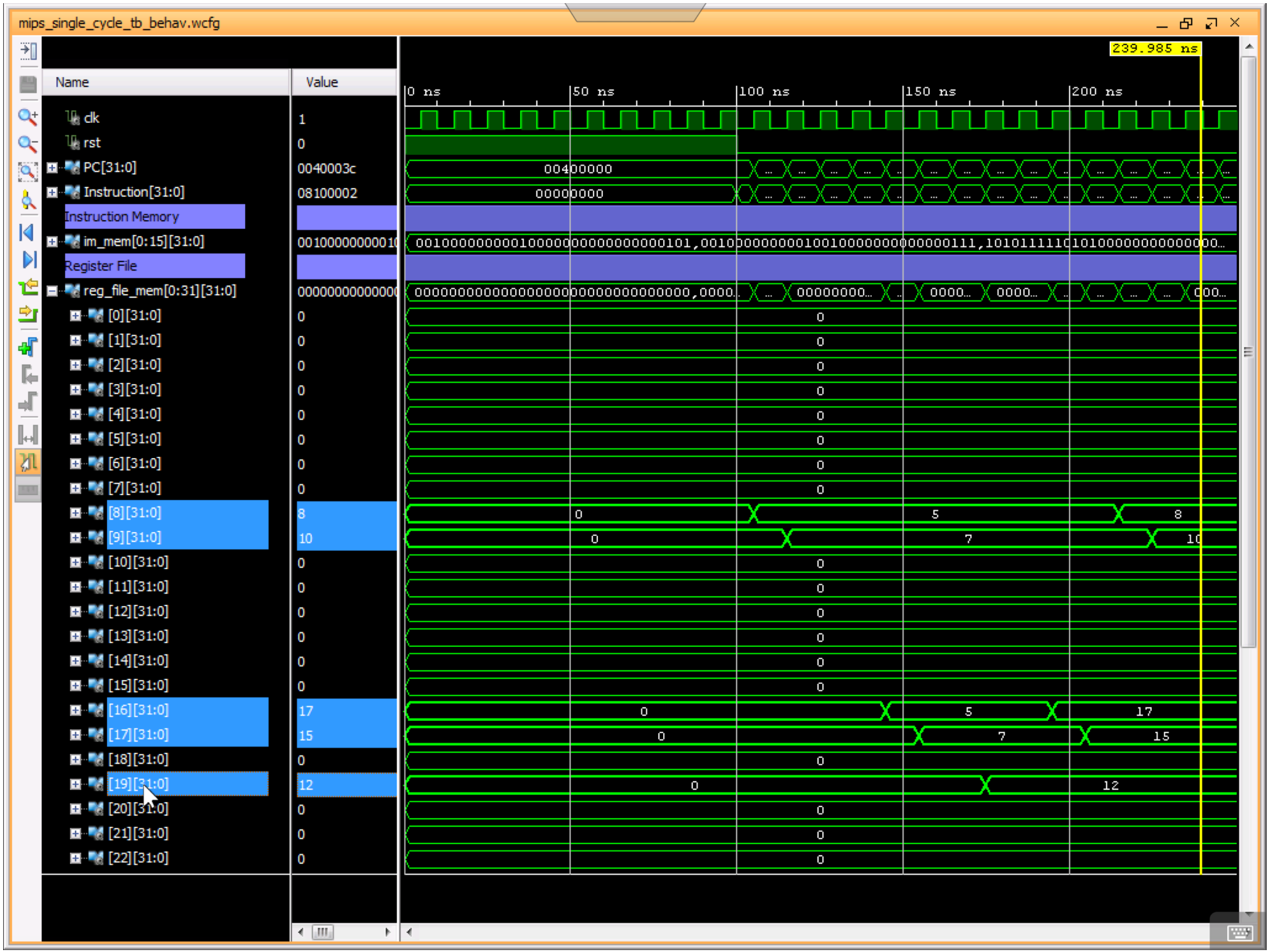
$DM_i = DM_0 + 4xD_i$
→ $D_i = (DM_i - DM_0)/4$

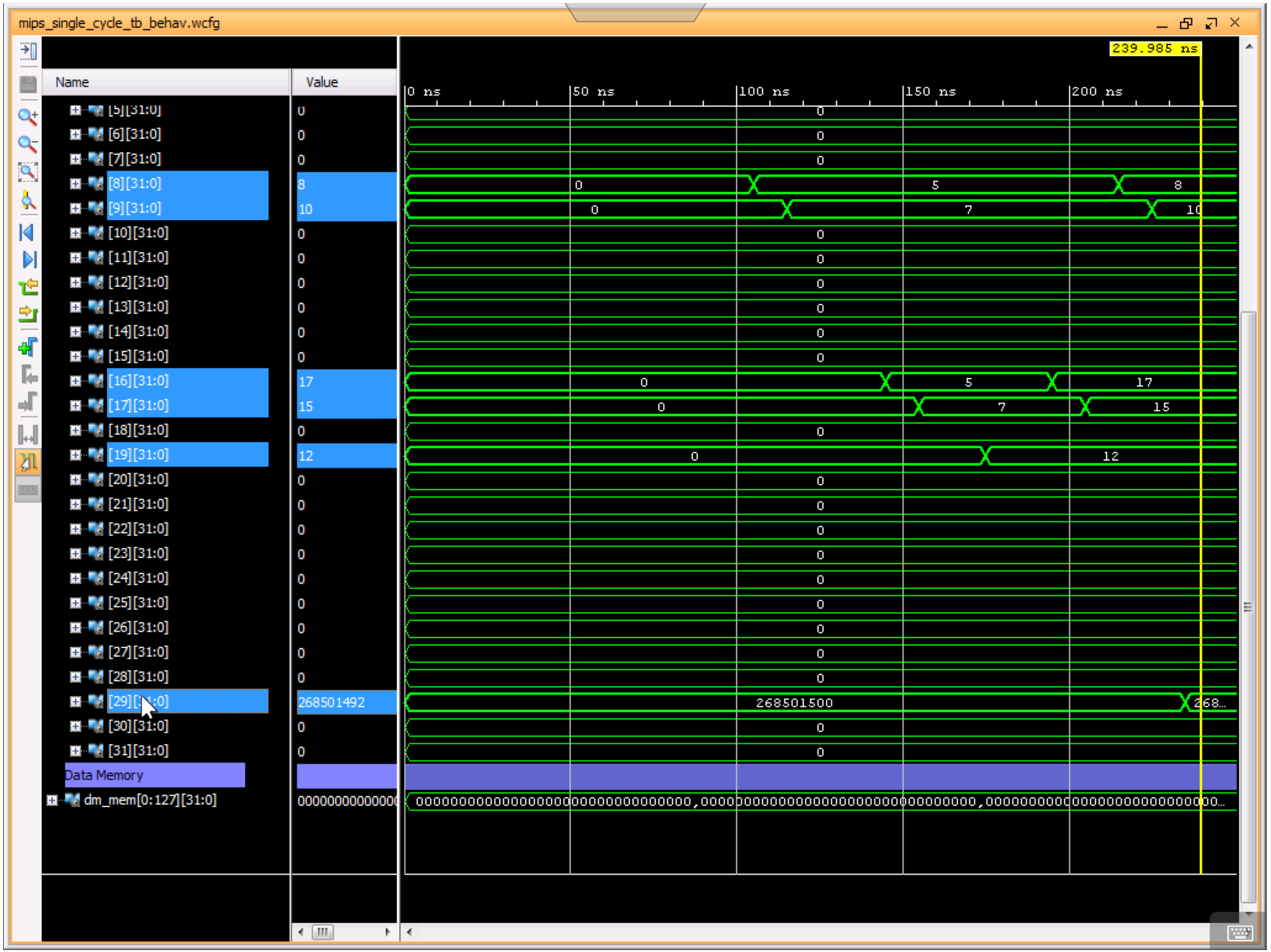
$DM_0 = A_0 = 0x1001\ 0000$, $DM_i = A_i$
→ $D_i = (A_i - A_0)/4$

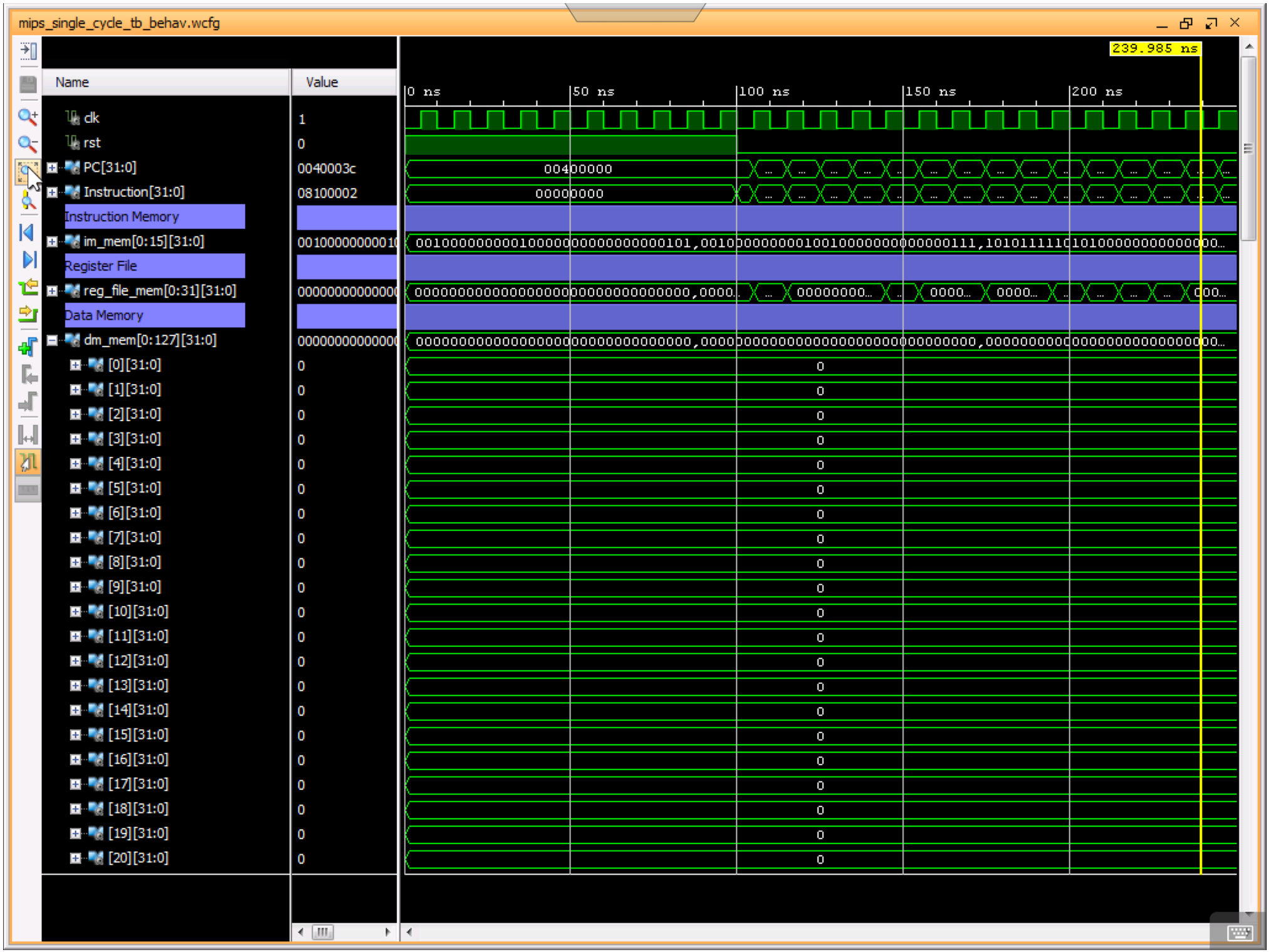
$\$sp_0 = DM_0 + 4x(Z_{DM}-1)$

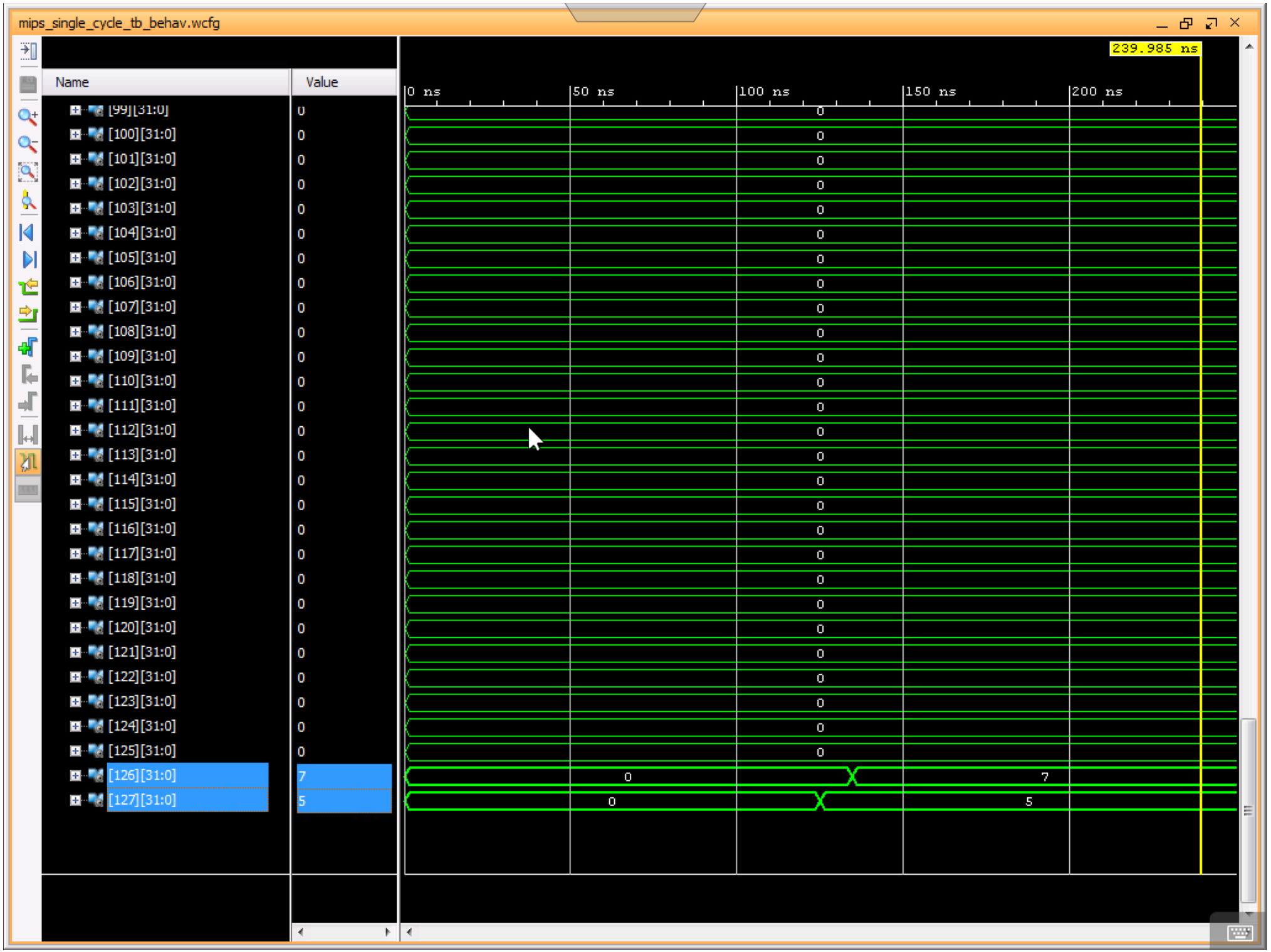
$DM_0 = A_0 = 0x1001\ 0000$
→ $\$sp_0 = A_0 + 4x(Z_{DM}-1)$


mips_single_cycle_tb_behav.wcfg
Name
Value
239.985 ns
0 ns
50 ns
100 ns
150 ns
200 ns
clk
1
rst
0
PC[31:0]
0040003c
Instruction[31:0]
08100002
Instruction Memory
im_mem[0:15][31:0]
[0][31:0]
[1][31:0]
[2][31:0]
[3][31:0]
[4][31:0]
[5][31:0]
[6][31:0]
[7][31:0]
[8][31:0]
[9][31:0]
[10][31:0]
[11][31:0]
[12][31:0]
[13][31:0]
[14][31:0]
[15][31:0]
Register File
reg_file_mem[0:31][31:0]
Data Memory
dm_mem[0:127][31:0]
00400000
00000000
0810000a
08100002


mips_single_cycle_tb_behav.wcfg
Name
Value
239.985 ns
0 ns
50 ns
100 ns
150 ns
200 ns
clk
1
rst
0
PC[31:0]
0040003c
004000000
Instruction[31:0]
08100002
00000000
Instruction Memory
im_mem[0:15][31:0]
Register File
reg_file_mem[0:31][31:0]
[0][31:0]
[1][31:0]
[2][31:0]
[3][31:0]
[4][31:0]
[5][31:0]
[6][31:0]
[7][31:0]
[8][31:0]
8
[9][31:0]
10
[10][31:0]
[11][31:0]
[12][31:0]
[13][31:0]
[14][31:0]
[15][31:0]
[16][31:0]
17
[17][31:0]
15
[18][31:0]
[19][31:0]
12
[20][31:0]
[21][31:0]
[22][31:0]


mips_single_cycle_tb_behav.wcfg
Name
Value
239.985 ns
0 ns
50 ns
100 ns
150 ns
200 ns
[5][31:0]
[6][31:0]
[7][31:0]
[8][31:0]
[9][31:0]
[10][31:0]
[11][31:0]
[12][31:0]
[13][31:0]
[14][31:0]
[15][31:0]
[16][31:0]
[17][31:0]
[18][31:0]
[19][31:0]
[20][31:0]
[21][31:0]
[22][31:0]
[23][31:0]
[24][31:0]
[25][31:0]
[26][31:0]
[27][31:0]
[28][31:0]
[29][31:0]
[30][31:0]
[31][31:0]
Data Memory
dm_mem[0:127][31:0]
268501492
268501500


mips_single_cycle_tb_behav.wcfg
Name
Value
239.985 ns
0 ns
50 ns
100 ns
150 ns
200 ns
clk
1
rst
0
PC[31:0]
0040003c
004000000
Instruction[31:0]
08100002
00000000
Instruction Memory
im_mem[0:15][31:0]
Register File
reg_file_mem[0:31][31:0]
Data Memory
dm_mem[0:127][31:0]
[0][31:0]
[1][31:0]
[2][31:0]
[3][31:0]
[4][31:0]
[5][31:0]
[6][31:0]
[7][31:0]
[8][31:0]
[9][31:0]
[10][31:0]
[11][31:0]
[12][31:0]
[13][31:0]
[14][31:0]
[15][31:0]
[16][31:0]
[17][31:0]
[18][31:0]
[19][31:0]
[20][31:0]


mips_single_cycle_tb_behav.wcfg
Name
Value
239.985 ns
0 ns
50 ns
100 ns
150 ns
200 ns
[100][31:0]
[101][31:0]
[102][31:0]
[103][31:0]
[104][31:0]
[105][31:0]
[106][31:0]
[107][31:0]
[108][31:0]
[109][31:0]
[110][31:0]
[111][31:0]
[112][31:0]
[113][31:0]
[114][31:0]
[115][31:0]
[116][31:0]
[117][31:0]
[118][31:0]
[119][31:0]
[120][31:0]
[121][31:0]
[122][31:0]
[123][31:0]
[124][31:0]
[125][31:0]
[126][31:0]
[127][31:0]
7
5