THE UNIVERSITY OF KANSAS

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

EECS 645 – Computer Architecture

Fall 2016

Final Project (Single-Cycle MIPS)

Student Name: Student ID:

Final Project

In this project you will be designing the Single-Cycle version of the MIPS processor that supports a subset, 13 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) 5 Control transfer: *beq, bne, j, jal, jr*

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 "Final_Project_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; "\Final_Project_MIPS_Single_Cycle\07_MIPS_Single_Cycle\design_sources" and "\Final_Project_MIPS_Single_Cycle\07_MIPS_Single_Cycle\simulation_sources" respectively.
4) Edit the VHDL files in the folder "\Final_Project_MIPS_Single_Cycle\07_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. ***100 µs***, 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:

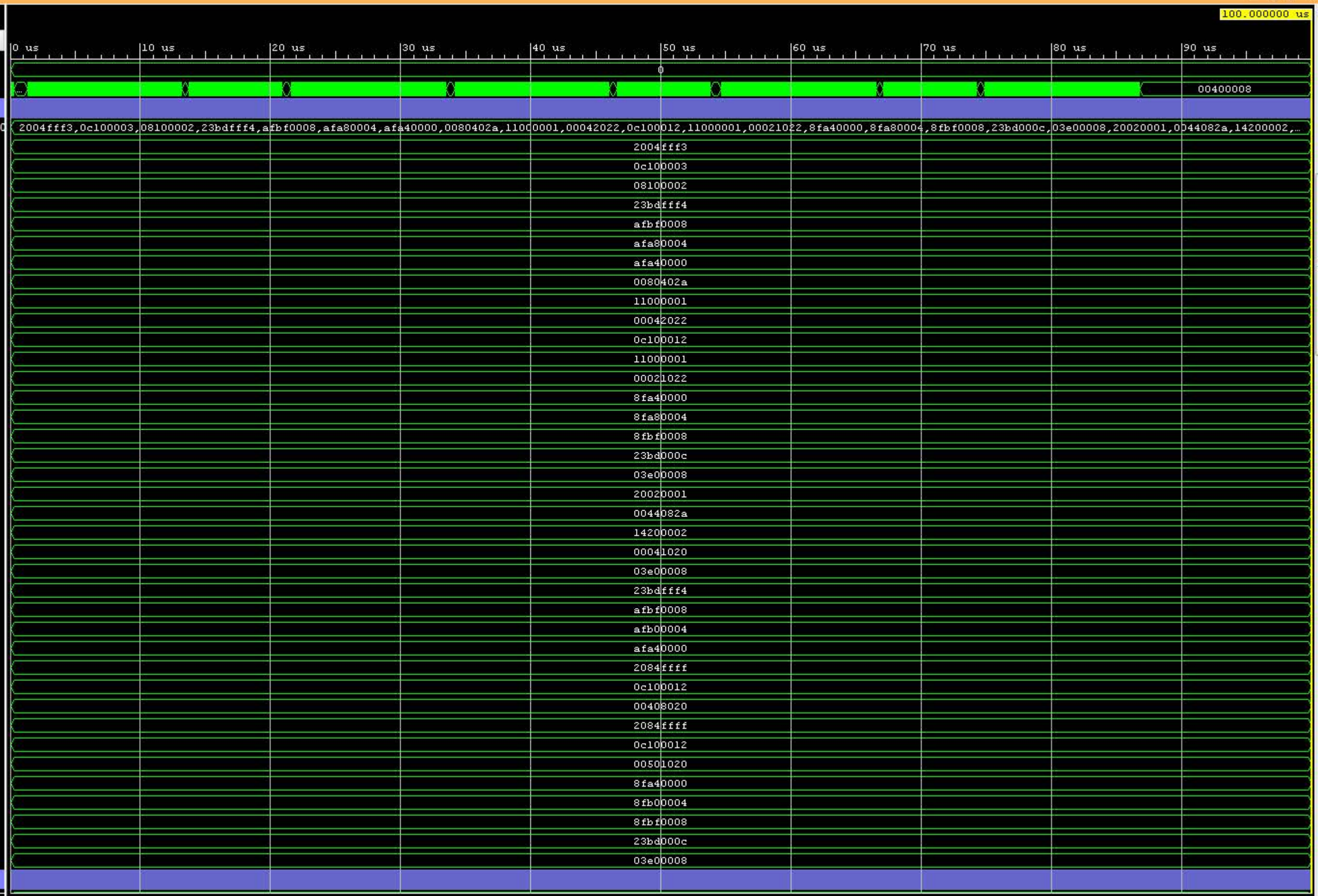
- Follow MIPS and MARS convention for the memory map as shown in the attached document
  - The code/text segment should start at address 0x00400000, and
  - The data segment should start at address 0x10010000
- You could use the provided assembly test program "\Final_Project_MIPS_Single_Cycle\07_MIPS_Single_Cycle\testing_options\fibonacci_recursive_pos_&_neg.asm" to verify your design by comparing your simulation results in Vivado with the run results of the same test program in MARS.
  - The proper simulation time for this test program should be set to a value larger than *87 µs*, e.g. ***100 µs.***

mips_single_cycle_wave.wcfg

Name	Value
clk	0
rst	0
PC[31:0]	00400008
Instruction[31:0]	08100002
Register File	
reg_file_mem[0:31][31:0]	00000000,00000
[0][31:0]	0
[1][31:0]	0
[2][31:0]	-233
[3][31:0]	0
[4][31:0]	-13
[5][31:0]	0
[6][31:0]	0
[7][31:0]	0
[8][31:0]	0
[9][31:0]	0
[10][31:0]	0
[11][31:0]	0
[12][31:0]	0
[13][31:0]	0
[14][31:0]	0
[15][31:0]	0
[16][31:0]	0
[17][31:0]	0
[18][31:0]	0
[19][31:0]	0
[20][31:0]	0
[21][31:0]	0
[22][31:0]	0
[23][31:0]	0
[24][31:0]	0
[25][31:0]	0
[26][31:0]	0
[27][31:0]	0
[28][31:0]	0
[29][31:0]	100101fc
[30][31:0]	0
[31][31:0]	00400008
Instruction Memory	
im_mem[0:37][31:0]	2004fff3,0c100
[0][31:0]	2004fff3
[1][31:0]	0c100003
[2][31:0]	08100002

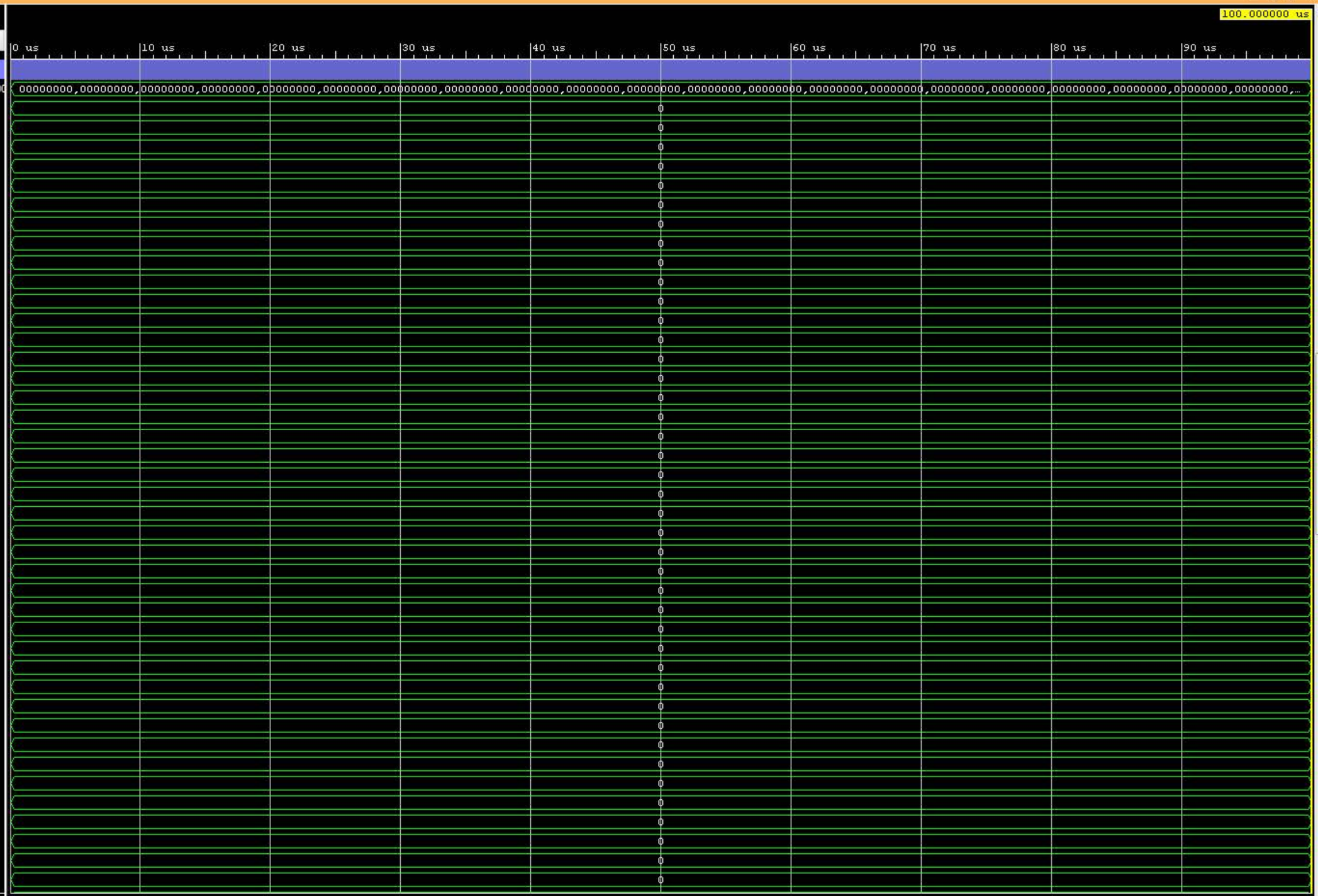
mips_single_cycle_wave.wcfg

Name	Value
[30][31:0]	0
[31][31:0]	00400008
Instruction Memory	
im_mem[0:37][31:0]	2004fff3,0c1000
[0][31:0]	2004fff3
[1][31:0]	0c100003
[2][31:0]	08100002
[3][31:0]	23bdfff4
[4][31:0]	afbf0008
[5][31:0]	afa80004
[6][31:0]	afa40000
[7][31:0]	0080402a
[8][31:0]	11000001
[9][31:0]	00042022
[10][31:0]	0c100012
[11][31:0]	11000001
[12][31:0]	00021022
[13][31:0]	8fa40000
[14][31:0]	8fa80004
[15][31:0]	8fbf0008
[16][31:0]	23bd000c
[17][31:0]	03e00008
[18][31:0]	20020001
[19][31:0]	0044082a
[20][31:0]	14200002
[21][31:0]	00041020
[22][31:0]	03e00008
[23][31:0]	23bdfff4
[24][31:0]	afbf0008
[25][31:0]	afb00004
[26][31:0]	afa40000
[27][31:0]	2084ffff
[28][31:0]	0c100012
[29][31:0]	00408020
[30][31:0]	2084ffff
[31][31:0]	0c100012
[32][31:0]	00501020
[33][31:0]	8fa40000
[34][31:0]	8fb00004
[35][31:0]	8fbf0008
[36][31:0]	23bd000c
[37][31:0]	03e00008
Data Memory	



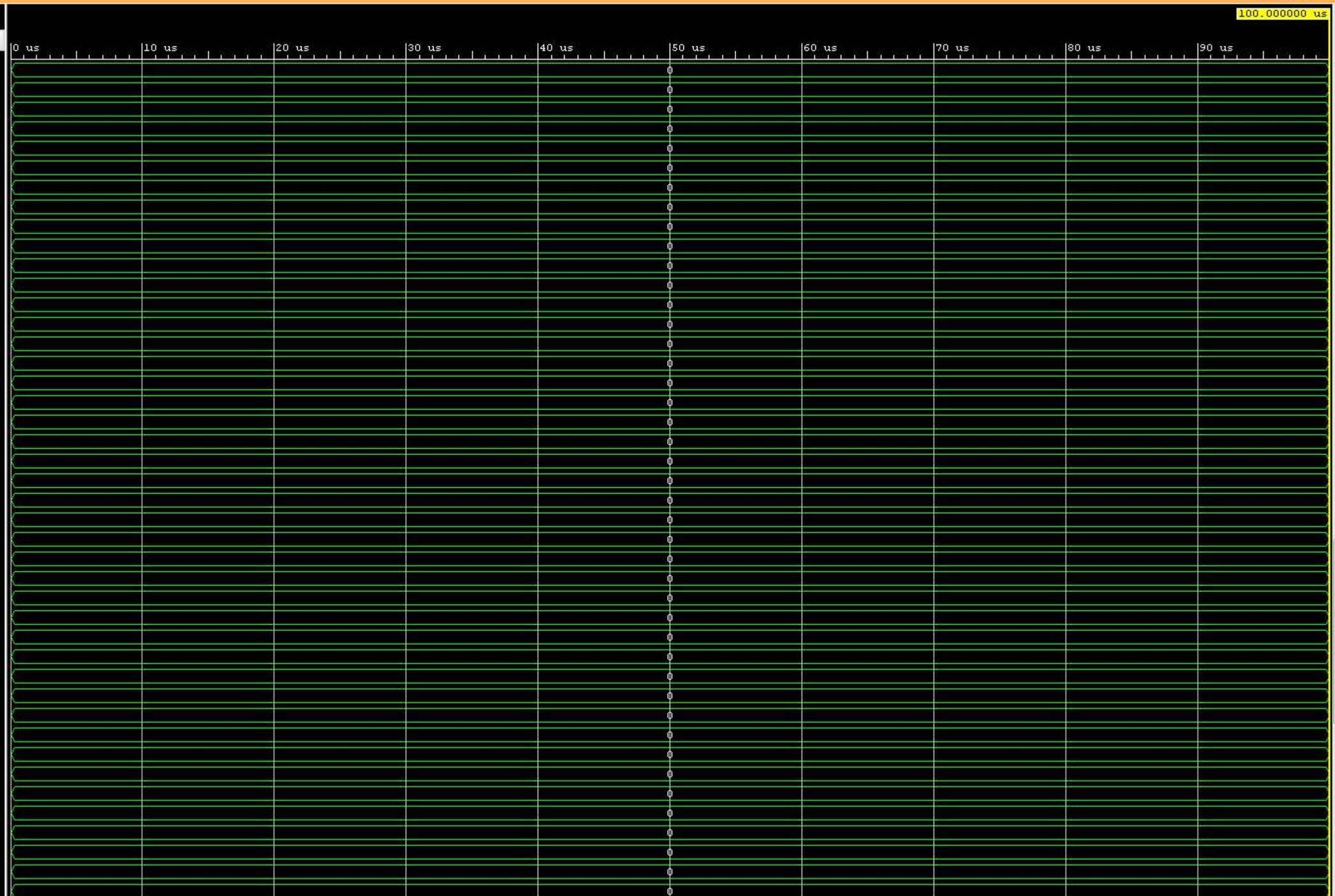
mips_single_cycle_wave.wcfg

Name	Value
Data Memory	
dm_mem[0:127][31:0]	00000000,00000
[0][31:0]	0
[1][31:0]	0
[2][31:0]	0
[3][31:0]	0
[4][31:0]	0
[5][31:0]	0
[6][31:0]	0
[7][31:0]	0
[8][31:0]	0
[9][31:0]	0
[10][31:0]	0
[11][31:0]	0
[12][31:0]	0
[13][31:0]	0
[14][31:0]	0
[15][31:0]	0
[16][31:0]	0
[17][31:0]	0
[18][31:0]	0
[19][31:0]	0
[20][31:0]	0
[21][31:0]	0
[22][31:0]	0
[23][31:0]	0
[24][31:0]	0
[25][31:0]	0
[26][31:0]	0
[27][31:0]	0
[28][31:0]	0
[29][31:0]	0
[30][31:0]	0
[31][31:0]	0
[32][31:0]	0
[33][31:0]	0
[34][31:0]	0
[35][31:0]	0
[36][31:0]	0
[37][31:0]	0
[38][31:0]	0
[39][31:0]	0
[40][31:0]	0



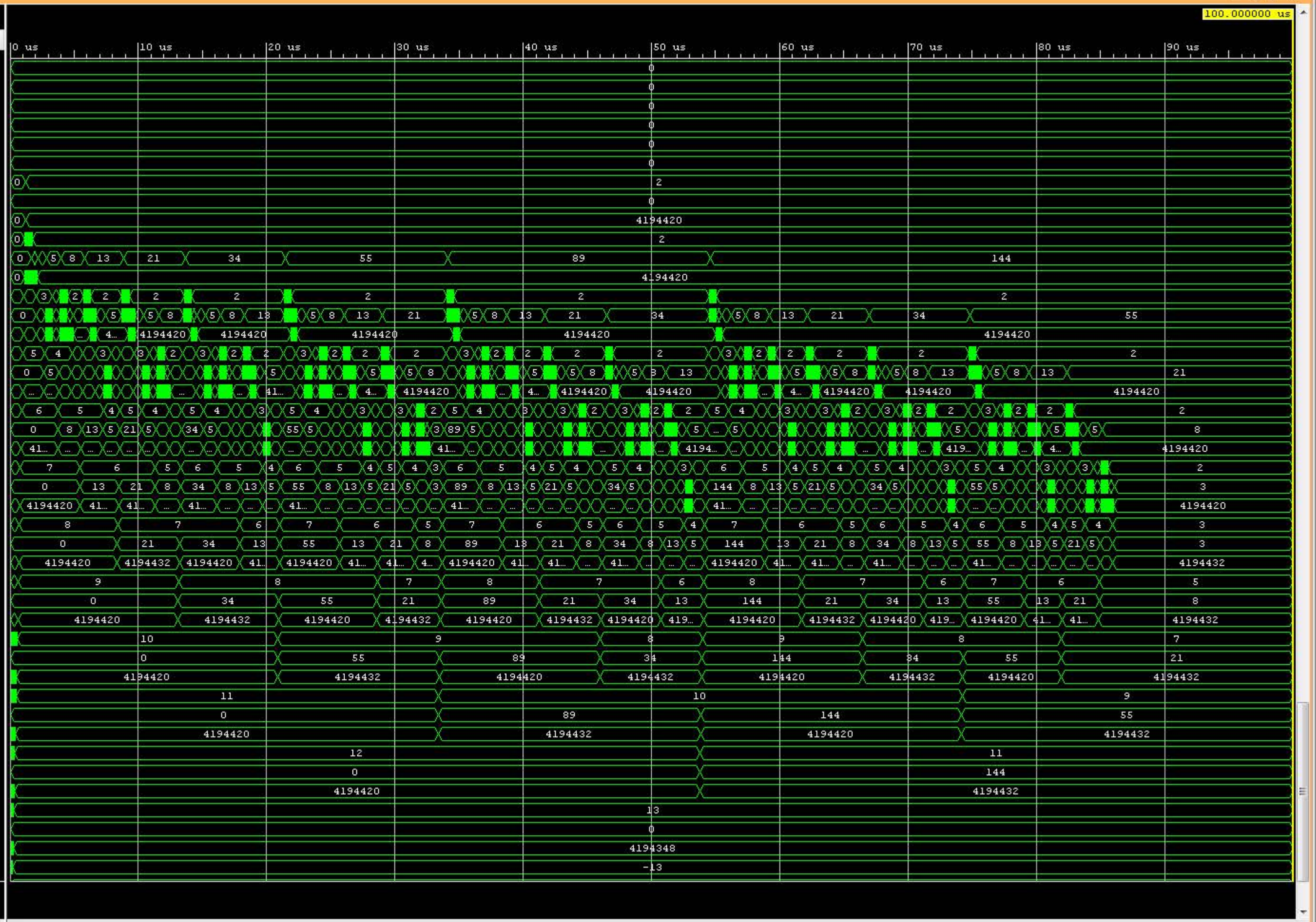
mips_single_cycle_wave.wcfg

Name	Value
[40][31:0]	0
[41][31:0]	0
[42][31:0]	0
[43][31:0]	0
[44][31:0]	0
[45][31:0]	0
[46][31:0]	0
[47][31:0]	0
[48][31:0]	0
[49][31:0]	0
[50][31:0]	0
[51][31:0]	0
[52][31:0]	0
[53][31:0]	0
[54][31:0]	0
[55][31:0]	0
[56][31:0]	0
[57][31:0]	0
[58][31:0]	0
[59][31:0]	0
[60][31:0]	0
[61][31:0]	0
[62][31:0]	0
[63][31:0]	0
[64][31:0]	0
[65][31:0]	0
[66][31:0]	0
[67][31:0]	0
[68][31:0]	0
[69][31:0]	0
[70][31:0]	0
[71][31:0]	0
[72][31:0]	0
[73][31:0]	0
[74][31:0]	0
[75][31:0]	0
[76][31:0]	0
[77][31:0]	0
[78][31:0]	0
[79][31:0]	0
[80][31:0]	0
[81][31:0]	0
[82][31:0]	0



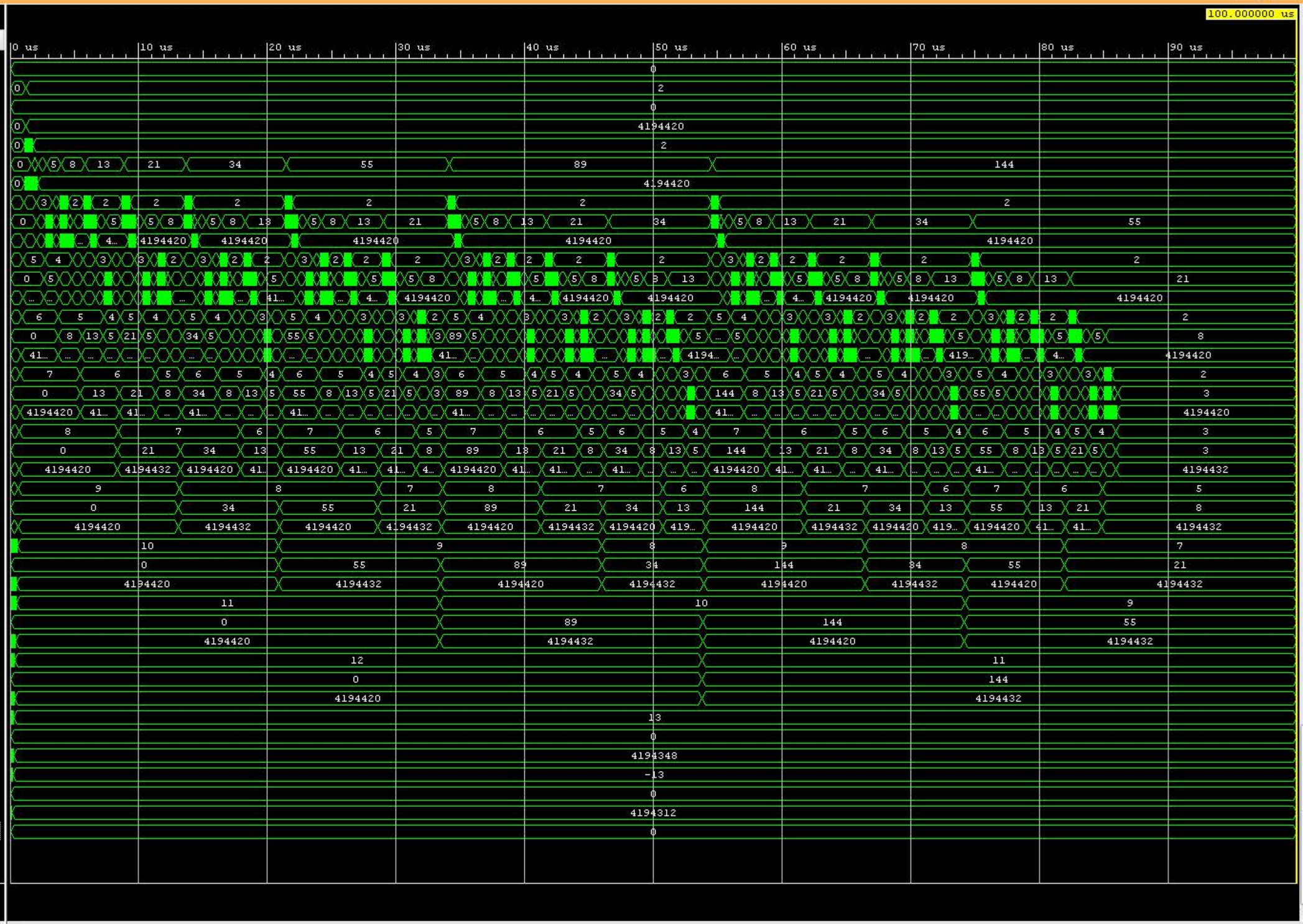
mips_single_cycle_wave.wcfg

Name	Value
[82][31:0]	0
[83][31:0]	0
[84][31:0]	0
[85][31:0]	0
[86][31:0]	0
[87][31:0]	0
[88][31:0]	2
[89][31:0]	0
[90][31:0]	4194420
[91][31:0]	2
[92][31:0]	144
[93][31:0]	4194420
[94][31:0]	2
[95][31:0]	55
[96][31:0]	4194420
[97][31:0]	2
[98][31:0]	21
[99][31:0]	4194420
[100][31:0]	2
[101][31:0]	8
[102][31:0]	4194420
[103][31:0]	2
[104][31:0]	3
[105][31:0]	4194420
[106][31:0]	3
[107][31:0]	3
[108][31:0]	4194432
[109][31:0]	5
[110][31:0]	8
[111][31:0]	4194432
[112][31:0]	7
[113][31:0]	21
[114][31:0]	4194432
[115][31:0]	9
[116][31:0]	55
[117][31:0]	4194432
[118][31:0]	11
[119][31:0]	144
[120][31:0]	4194432
[121][31:0]	13
[122][31:0]	0
[123][31:0]	4194348
[124][31:0]	-13



mips_single_cycle_wave.wcfg

Name	Value
[87][31:0]	0
[88][31:0]	2
[89][31:0]	0
[90][31:0]	4194420
[91][31:0]	2
[92][31:0]	144
[93][31:0]	4194420
[94][31:0]	2
[95][31:0]	55
[96][31:0]	4194420
[97][31:0]	2
[98][31:0]	21
[99][31:0]	4194420
[100][31:0]	2
[101][31:0]	8
[102][31:0]	4194420
[103][31:0]	2
[104][31:0]	3
[105][31:0]	4194420
[106][31:0]	3
[107][31:0]	3
[108][31:0]	4194432
[109][31:0]	5
[110][31:0]	8
[111][31:0]	4194432
[112][31:0]	7
[113][31:0]	21
[114][31:0]	4194432
[115][31:0]	9
[116][31:0]	55
[117][31:0]	4194432
[118][31:0]	11
[119][31:0]	144
[120][31:0]	4194432
[121][31:0]	13
[122][31:0]	0
[123][31:0]	4194348
[124][31:0]	-13
[125][31:0]	0
[126][31:0]	4194312
[127][31:0]	0

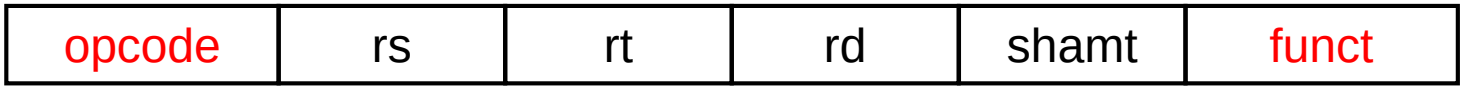


Extended Main Control Unit

Instr. OP	RegDst	ALUSrc	MemToReg	RegWr	MemRd	MemWr	Beq	Bne	ALUOp	J	Jal	Jr
R-type 000000 & /= 001000	1	0	0	1	0	0	0	0	10	0		
jr 000000 & 001000												
lw 100011	0	1	1	1	1	0	0	0	00	0		
sw 101011	0	1	0	0	0	1	0	0	00	0		
beq 000100	0	0	0	0	0	0	1	0	01	0		
bne 000101												
j 000010	0	0	0	0	0	0	0	0	00	1		
jal 000011												
addi 001000	0	1	0	1	0	0	0	0	00	0		

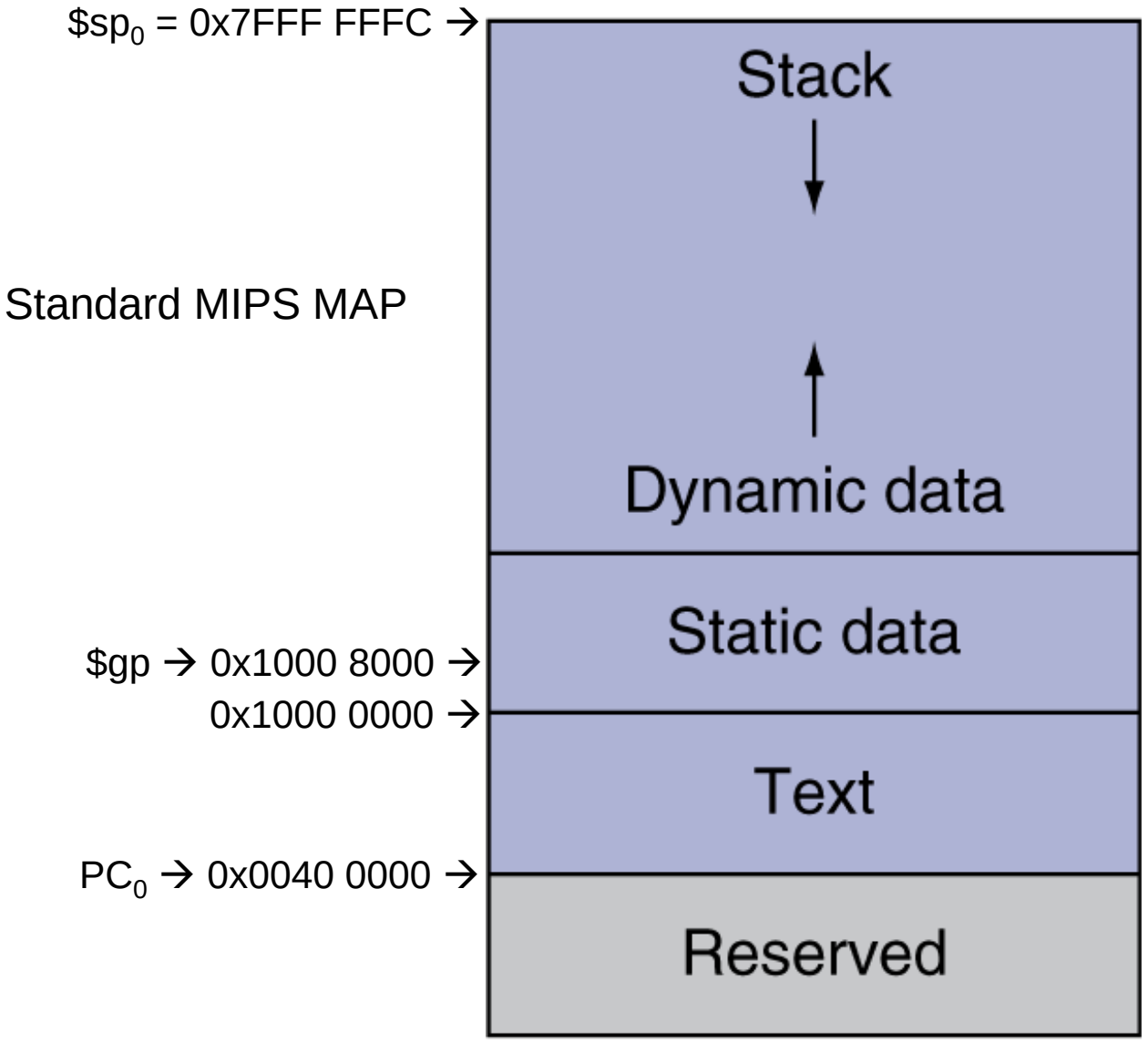
Extended ALU Control

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



opcode	ALUOp	Operation	funct	ALU function	ALU control
lw ≡ 100011	00	load word	XXXXXX	add	0010
sw ≡ 101011	00	store word	XXXXXX	add	0010
addi ≡ 001000	00	add immediate	XXXXXX	add	0010
beq ≡ 000100	01	branch equal	XXXXXX	subtract	0110
bne ≡ 000101	01	branch not 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

MIPS Memory Layout/Map



MIPS Memory Layout/Map

sp_0 = 0x7FFF FFFC →

sp_0 = 0x7FFF EFFC →

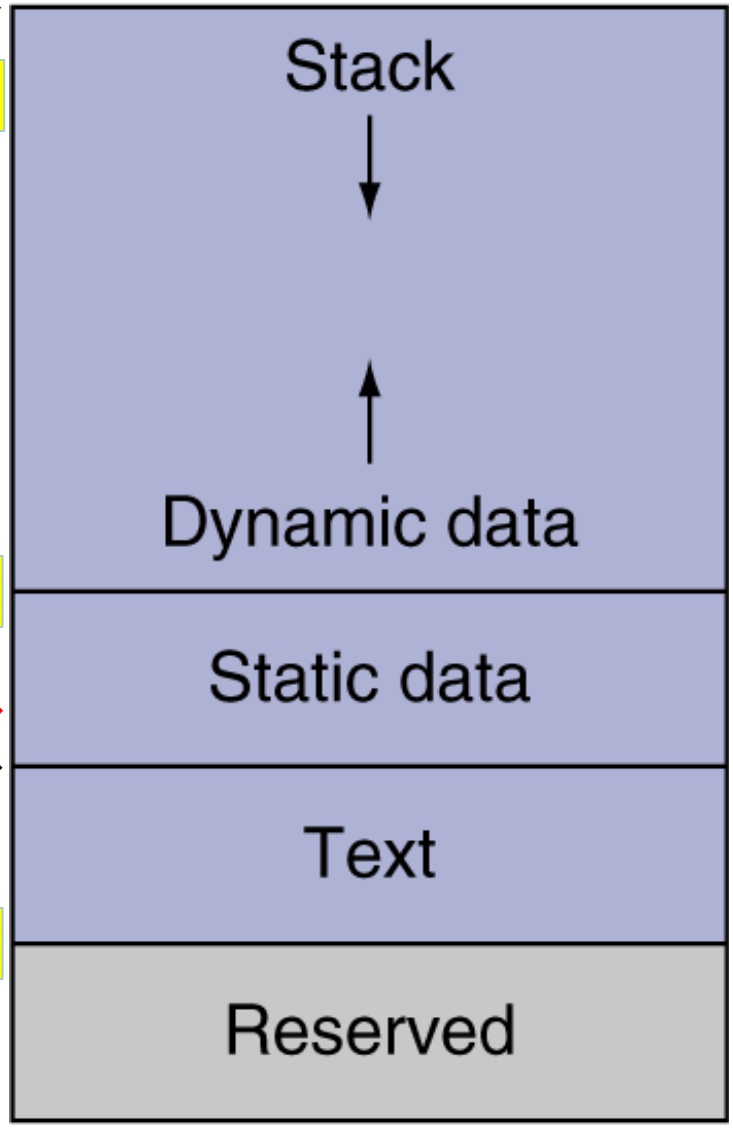
MIPS MAP in MARS

.data → 0x1001 0000 →

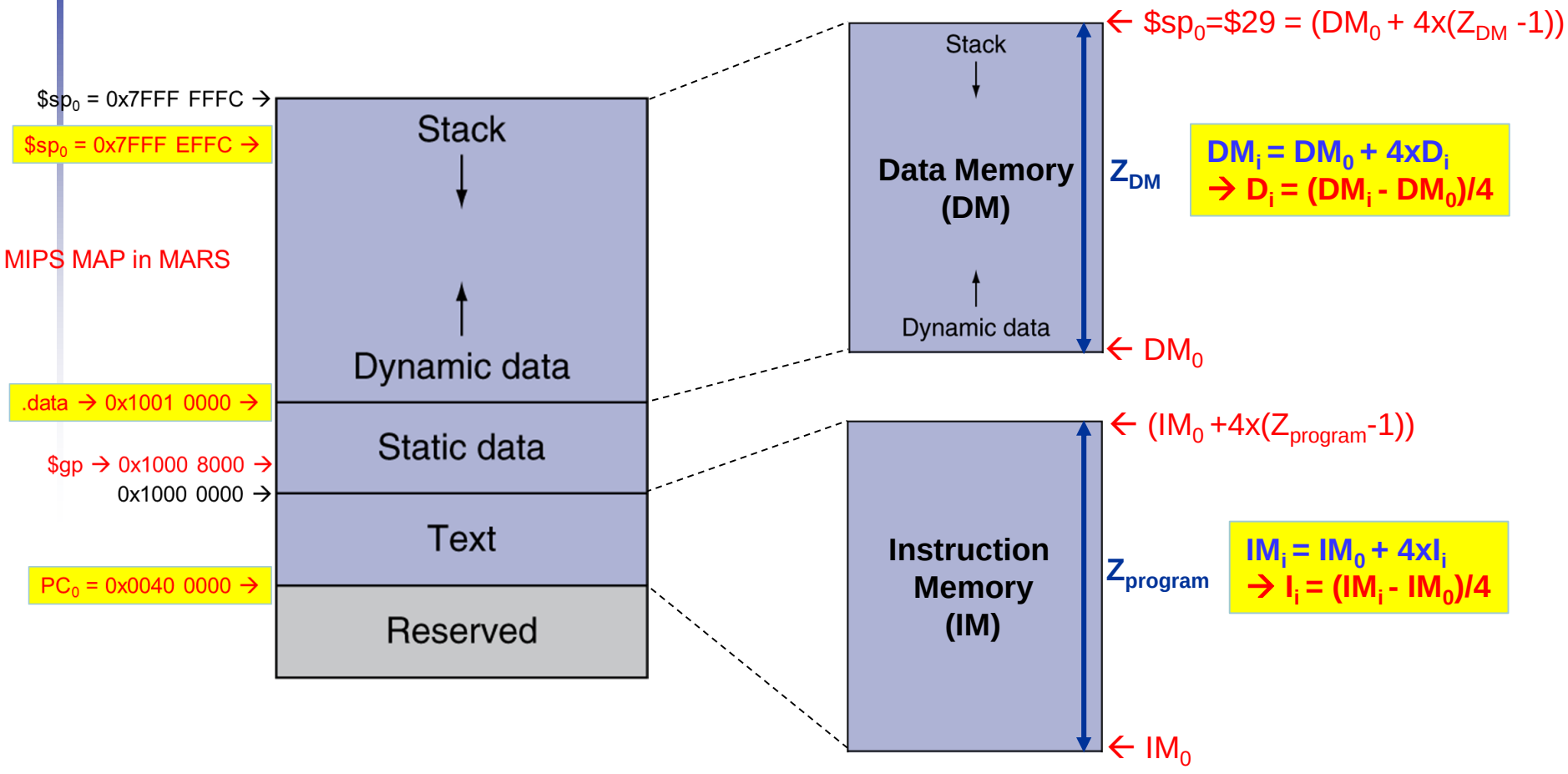
$gp → 0x1000 8000 →

0x1000 0000 →

PC_0 = 0x0040 0000 →



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**

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
```

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
→ $\$sp_0 = A_0 + 4x(Z_{DM}-1)$

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

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

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

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

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

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

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