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. Modify the provided microarchitecture datapath and control path shown in the following pages such that 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) Neatly regenerate/redraw the microarchitecture diagram to include your modifications,
b) Write the equivalent **VHDL code**, and
c) Verify the correct operation through Vivado Simulator by comparing your simulation results with those of MARS runs.

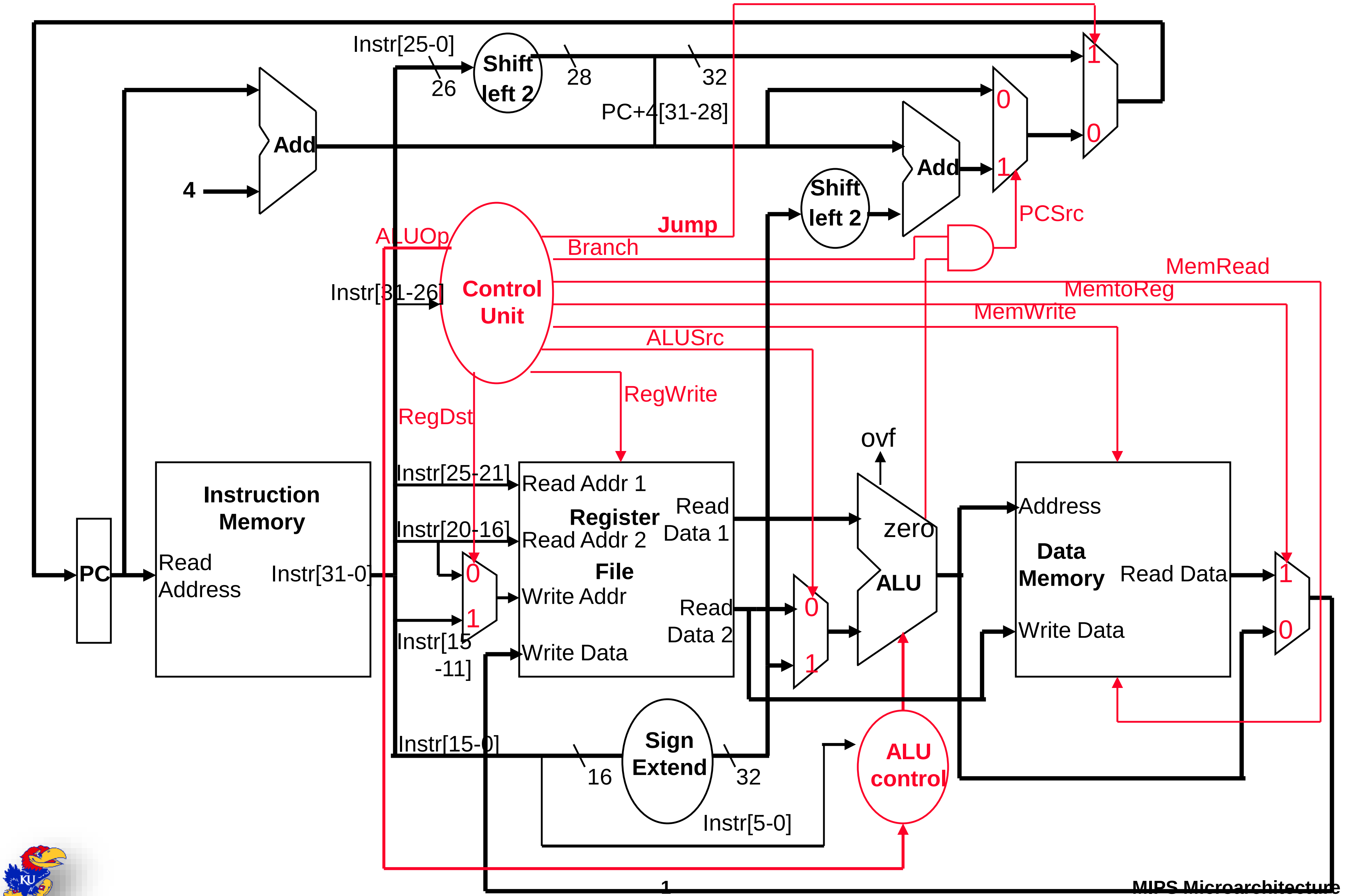
Steps:

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.***

Complete Single-Cycle/Non-Pipelined Datapath



MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

①

MIPS Reference Data

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+4;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rt] >>> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23_{hex}

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0
I	opcode	rs	rt	immediate		
	31 26	25 21	20 16	15 0		
J	opcode	address				
	31 26	25 0				

ARITHMETIC CORE INSTRUCTION SET

②

NAME, MNEMONIC		FOR-MAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	c.*x*.s*	FR	FPcond = (F[fs] *op* F[ft]) ? 1 : 0		11/10/--/*y*
FP Compare Double	c.*x*.d*	FR	FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/*y*
* (*x* is eq, lt, or le) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)					
FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi		0 /--/--/10
Move From Lo	mflo	R	R[rd] = Lo		0 /--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]		10 /0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >> shamt		0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31 26	25 21	20 16	15 11	10 6	5 0
FI	opcode	fmt	ft	immediate		
	31 26	25 21	20 16	15 0		

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

③

OPCODES, BASE CONVERSION, ASCII SYMBOLS

MIPS opcode (31:26)	(1) MIPS funct (5:0)	(2) MIPS funct (5:0)	Binary	Deci-mal	Hexa-deci-mal	ASCII Char-acter	Deci-mal	Hexa-deci-mal	ASCII Char-acter
(1)	sll	add.*f*	00 0000	0	0	NUL	64	40	@
		sub.*f*	00 0001	1	1	SOH	65	41	A
j	srl	mul.*f*	00 0010	2	2	STX	66	42	B
jal	sra	div.*f*	00 0011	3	3	ETX	67	43	C
beq	sllv	sqrt.*f*	00 0100	4	4	EOT	68	44	D
bne		abs.*f*	00 0101	5	5	ENQ	69	45	E
blez	srlv	mov.*f*	00 0110	6	6	ACK	70	46	F
bgtz	srav	neg.*f*	00 0111	7	7	BEL	71	47	G
addi	jr		00 1000	8	8	BS	72	48	H
addiu	jalr		00 1001	9	9	HT	73	49	I
slti	movz		00 1010	10	a	LF	74	4a	J
sltiu	movn		00 1011	11	b	VT	75	4b	K
andi	syscall	round.w.*f*	00 1100	12	c	FF	76	4c	L
ori	break	trunc.w.*f*	00 1101	13	d	CR	77	4d	M
xori		ceil.w.*f*	00 1110	14	e	SO	78	4e	N
lui	sync	floor.w.*f*	00 1111	15	f	SI	79	4f	O
	mfhi		01 0000	16	10	DLE	80	50	P
(2)	mthi		01 0001	17	11	DC1	81	51	Q
	mflo	movz.*f*	01 0010	18	12	DC2	82	52	R
	mtlo	movn.*f*	01 0011	19	13	DC3	83	53	S
			01 0100	20	14	DC4	84	54	T
			01 0101	21	15	NAK	85	55	U
			01 0110	22	16	SYN	86	56	V
			01 0111	23	17	ETB	87	57	W
	mult		01 1000	24	18	CAN	88	58	X
	multu		01 1001	25	19	EM	89	59	Y
	div		01 1010	26	1a	SUB	90	5a	Z
	divu		01 1011	27	1b	ESC	91	5b	[
			01 1100	28	1c	FS	92	5c	\
			01 1101	29	1d	GS	93	5d	]
			01 1110	30	1e	RS	94	5e	^
			01 1111	31	1f	US	95	5f	_
lb	add	cvt.s.*f*	10 0000	32	20	Space	96	60	‘
lh	addu	cvt.d.*f*	10 0001	33	21	!	97	61	a
lwl	sub		10 0010	34	22	"	98	62	b
lw	subu		10 0011	35	23	#	99	63	c
lbu	and	cvt.w.*f*	10 0100	36	24	$	100	64	d
lhu	or		10 0101	37	25	%	101	65	e
lwr	xor		10 0110	38	26	&	102	66	f
	nor		10 0111	39	27	’	103	67	g
sb			10 1000	40	28	(	104	68	h
sh			10 1001	41	29	)	105	69	i
swl	slt		10 1010	42	2a	*	106	6a	j
sw	sltu		10 1011	43	2b	+	107	6b	k
			10 1100	44	2c	,	108	6c	l
			10 1101	45	2d	-	109	6d	m
swr			10 1110	46	2e	.	110	6e	n
cache			10 1111	47	2f	/	111	6f	o
ll	tge	c.f.*f*	11 0000	48	30	0	112	70	p
lwc1	tgeu	c.un.*f*	11 0001	49	31	1	113	71	q
lwc2	tlt	c.eq.*f*	11 0010	50	32	2	114	72	r
pref	tltu	c.ueq.*f*	11 0011	51	33	3	115	73	s
	teq	c.olt.*f*	11 0100	52	34	4	116	74	t
ldc1		c.ult.*f*	11 0101	53	35	5	117	75	u
ldc2	tne	c.ole.*f*	11 0110	54	36	6	118	76	v
		c.ule.*f*	11 0111	55	37	7	119	77	w
sc		c.sf.*f*	11 1000	56	38	8	120	78	x
swc1		c.ngle.*f*	11 1001	57	39	9	121	79	y
swc2		c.seq.*f*	11 1010	58	3a	:	122	7a	z
		c.ngl.*f*	11 1011	59	3b	;	123	7b	{
		c.lt.*f*	11 1100	60	3c	<	124	7c	|
sdc1		c.nge.*f*	11 1101	61	3d	=	125	7d	}
sdc2		c.le.*f*	11 1110	62	3e	>	126	7e	~
		c.ngt.*f*	11 1111	63	3f	?	127	7f	DEL

(1) opcode(31:26) == 0

(2) opcode(31:26) == 17_{ten} (11_{hex}); if fmt(25:21)==16_{ten} (10_{hex}) *f* = s (single);
if fmt(25:21)==17_{ten} (11_{hex}) *f* = d (double)

④

IEEE 754 FLOATING-POINT STANDARD

$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent - Bias})}$$

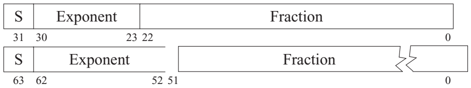
where Single Precision Bias = 127,
Double Precision Bias = 1023.

IEEE 754 Symbols

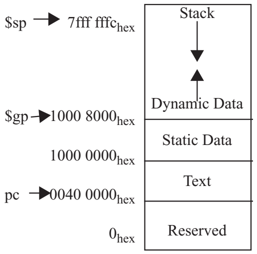
Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

S.P. MAX = 255, D.P. MAX = 2047

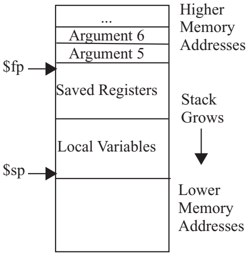
IEEE Single Precision and Double Precision Formats:



MEMORY ALLOCATION



STACK FRAME

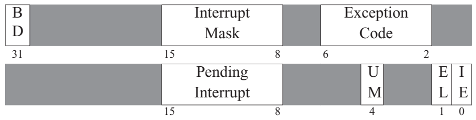


DATA ALIGNMENT

<table>
<tr><td colspan="8">Double Word</td></tr>
<tr><td colspan="4">Word</td><td colspan="4">Word</td></tr>
<tr><td colspan="2">Halfword</td><td colspan="2">Halfword</td><td colspan="2">Halfword</td><td colspan="2">Halfword</td></tr>
<tr><td>Byte</td><td>Byte</td><td>Byte</td><td>Byte</td><td>Byte</td><td>Byte</td><td>Byte</td><td>Byte</td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr>
</table>

Value of three least significant bits of byte address (Big Endian)

EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS



BD = Branch Delay, UM = User Mode, EL = Exception Level, IE =Interrupt Enable

EXCEPTION CODES

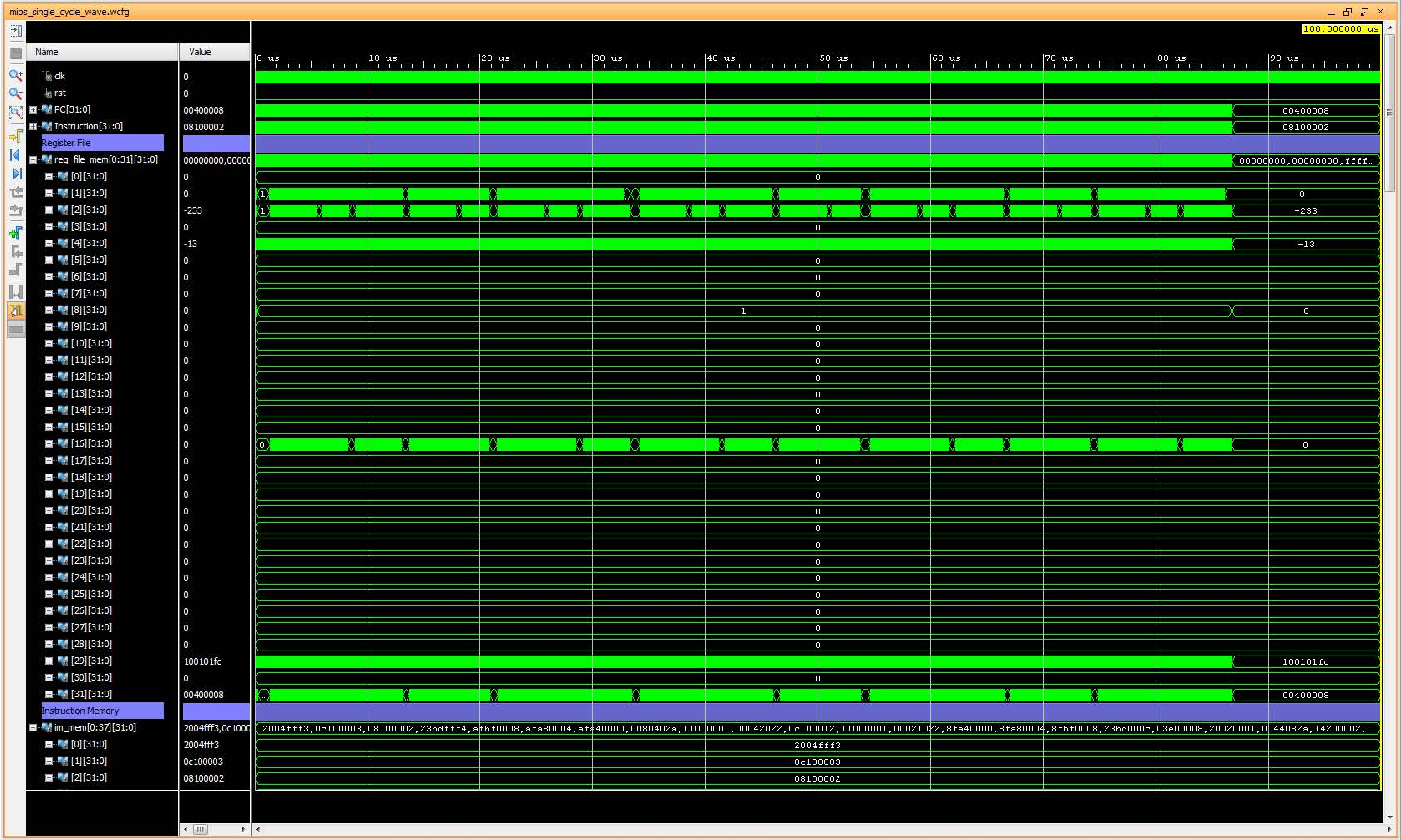
Number	Name	Cause of Exception	Number	Name	Cause of Exception
0	Int	Interrupt (hardware)	9	Bp	Breakpoint Exception
4	AdEL	Address Error Exception (load or instruction fetch)	10	RI	Reserved Instruction Exception
5	AdES	Address Error Exception (store)	11	CpU	Coprocessor Unimplemented
6	IBE	Bus Error on Instruction Fetch	12	Ov	Arithmetic Overflow Exception
7	DBE	Bus Error on Load or Store	13	Tr	Trap
8	Sys	Syscall Exception	15	FPE	Floating Point Exception

SIZE PREFIXES (10^x for Disk, Communication; 2^x for Memory)

SIZE	PRE-FIX	SIZE	PRE-FIX	SIZE	PRE-FIX	SIZE	PRE-FIX
10^3, 2^{10}	Kilo-	10^{15}, 2^{50}	Peta-	10^{-3}	milli-	10^{-15}	femto-
10^6, 2^{20}	Mega-	10^{18}, 2^{60}	Exa-	10^{-6}	micro-	10^{-18}	atto-
10^9, 2^{30}	Giga-	10^{21}, 2^{70}	Zetta-	10^{-9}	nano-	10^{-21}	zepto-
10^{12}, 2^{40}	Tera-	10^{24}, 2^{80}	Yotta-	10^{-12}	pico-	10^{-24}	yocto-

The symbol for each prefix is just its first letter, except µ is used for micro.

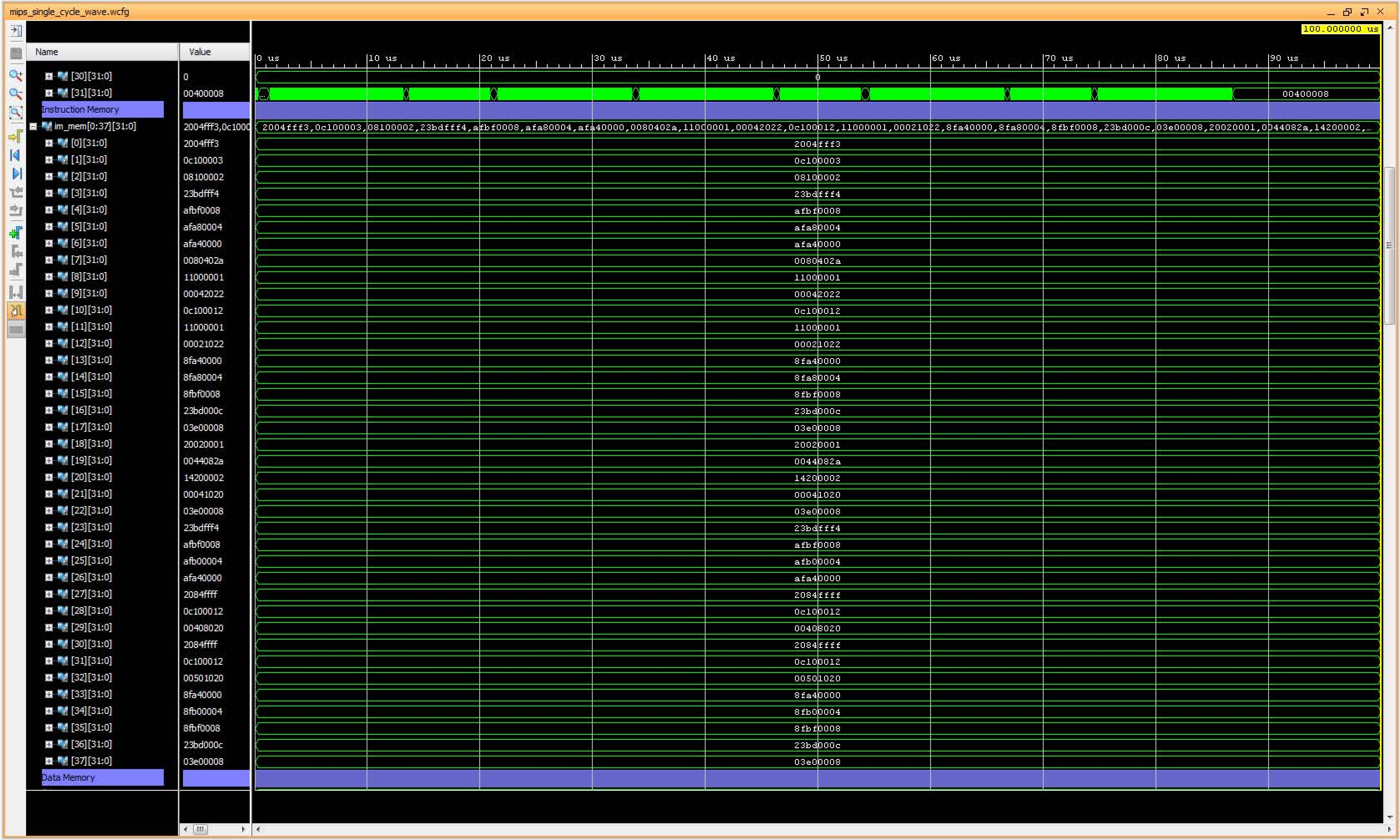
MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together



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]	2004fff3,0c1000
[0][31:0]	2004fff3
[1][31:0]	0c100003
[2][31:0]	08100002

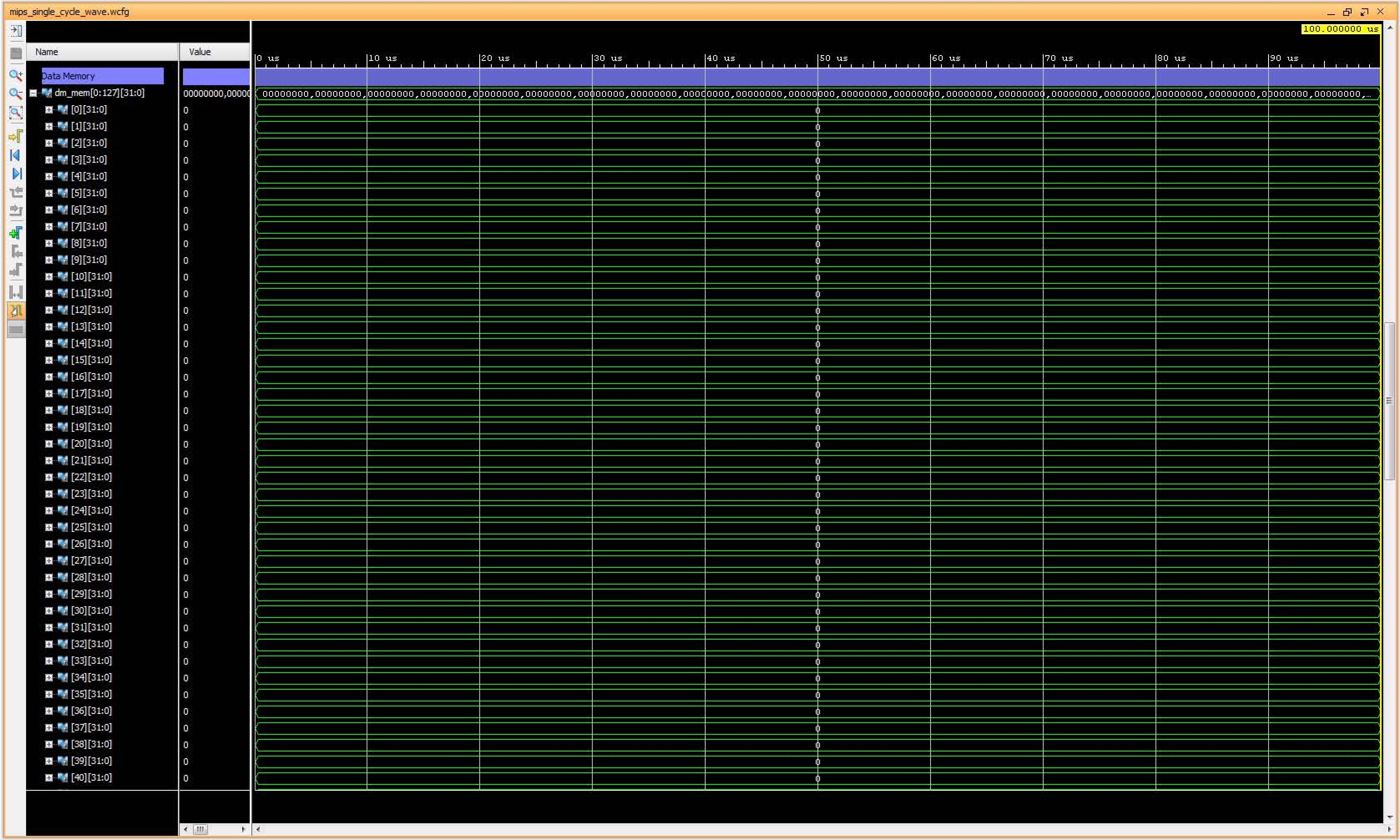
2004fff3,0c100003,08100002,23bdfff4,afbf0008,afa80004,afa40000,0080402a,11000001,00042022,0c100012,11000001,00021022,8fa40000,8fa80004,8fbf0008,23bd000c,03e00008,20020001,0044082a,14200002,…



mips_single_cycle_wave.wcfg

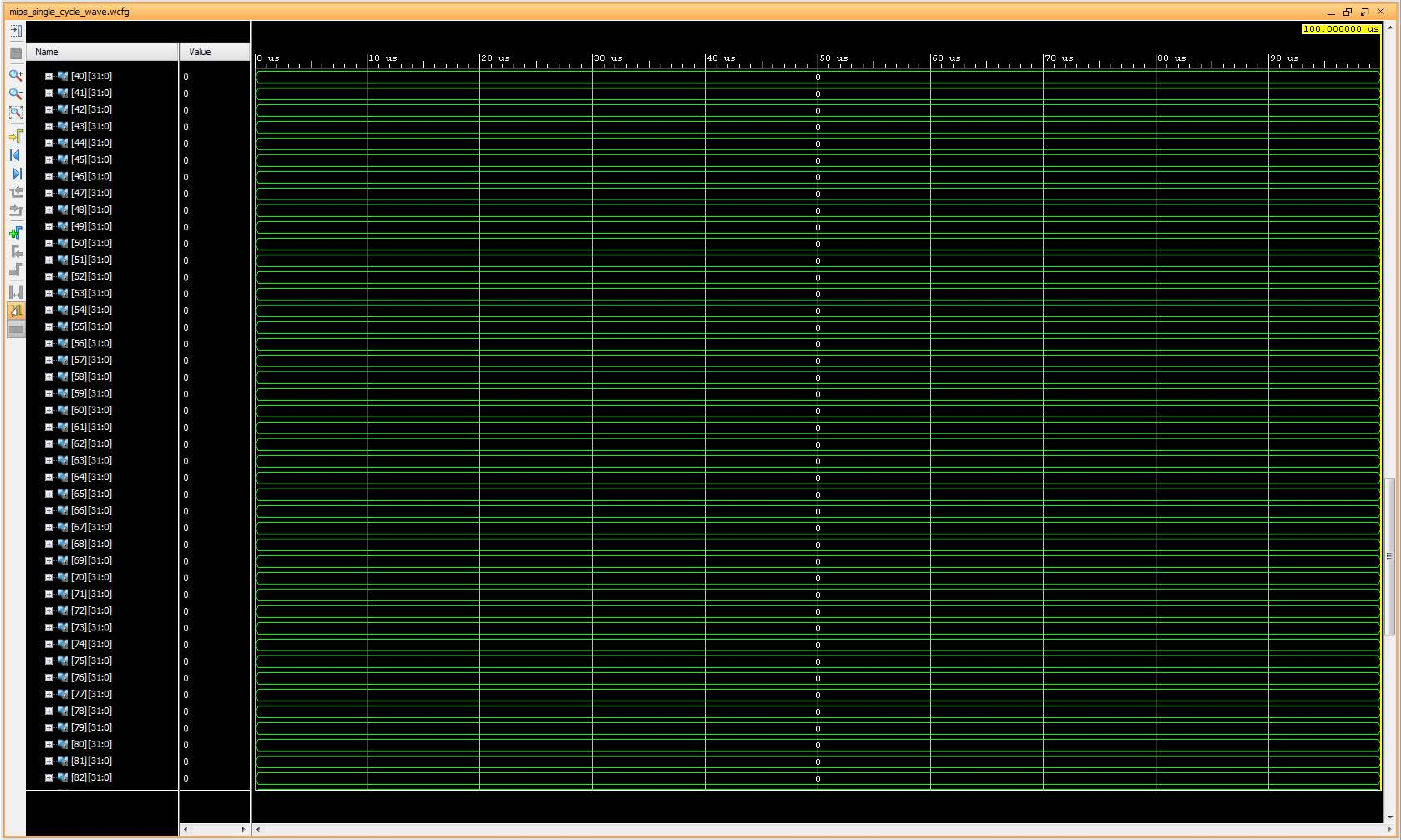
```
100.000000 us

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
```



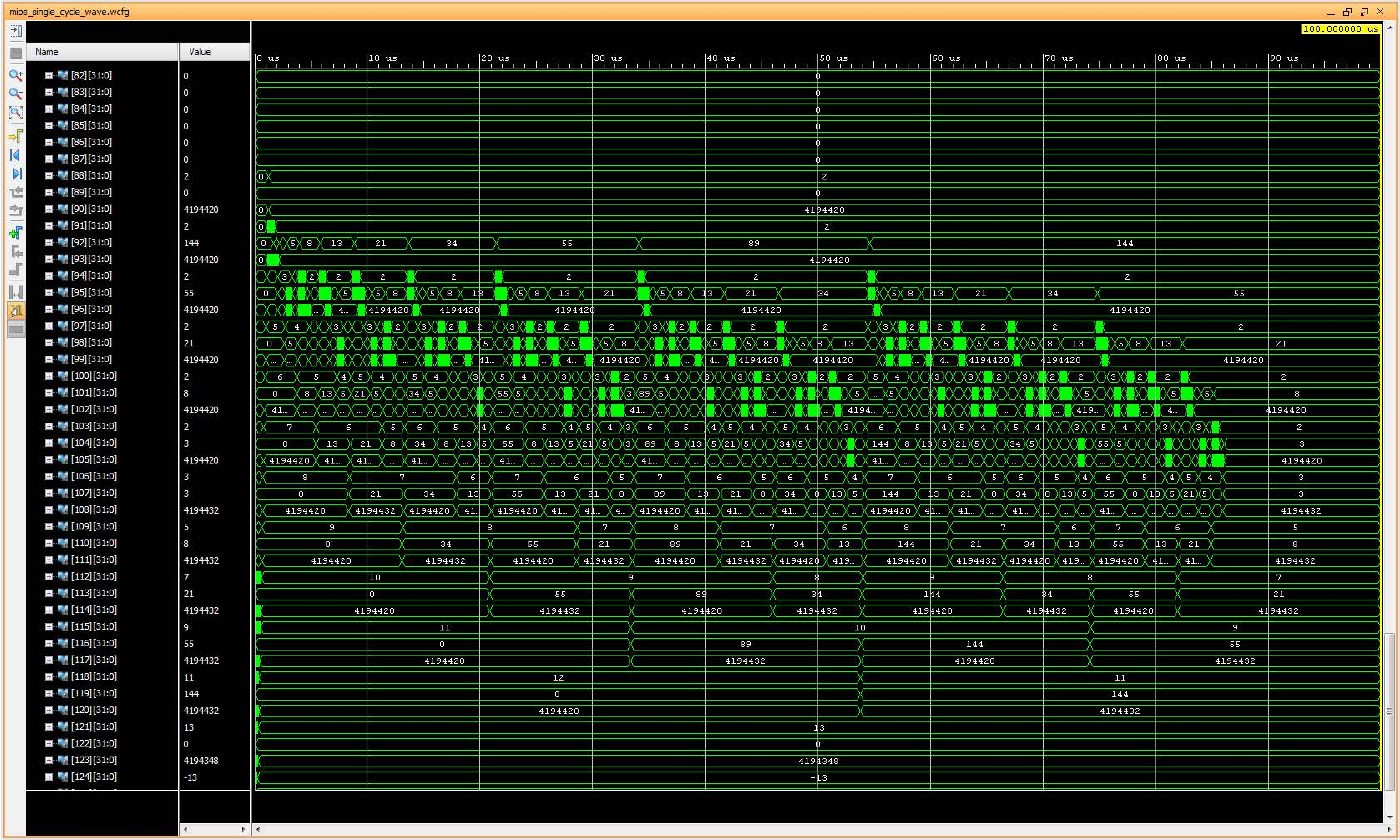
mips_single_cycle_wave.wcfg

```
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
```

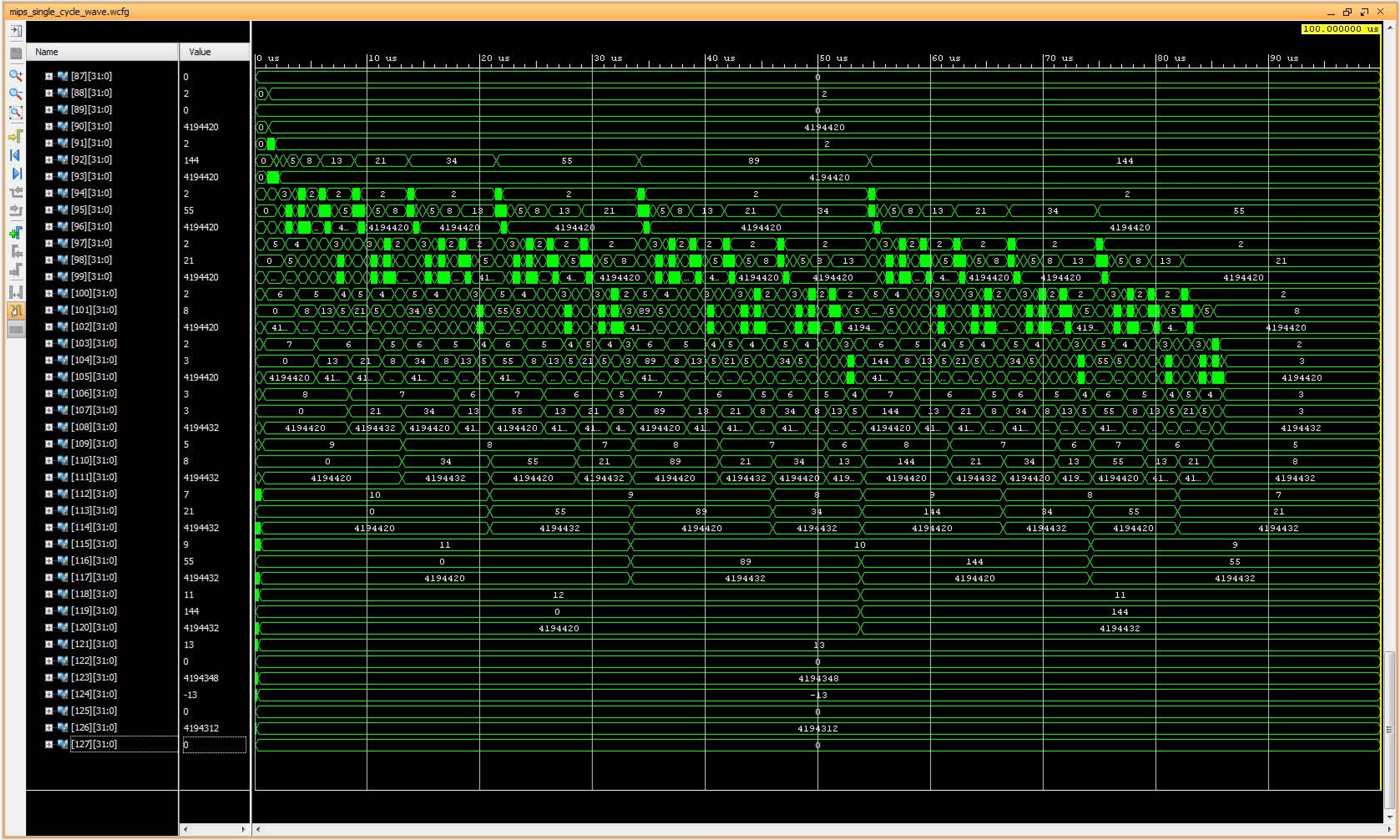


mips_single_cycle_wave.wcfg

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



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		

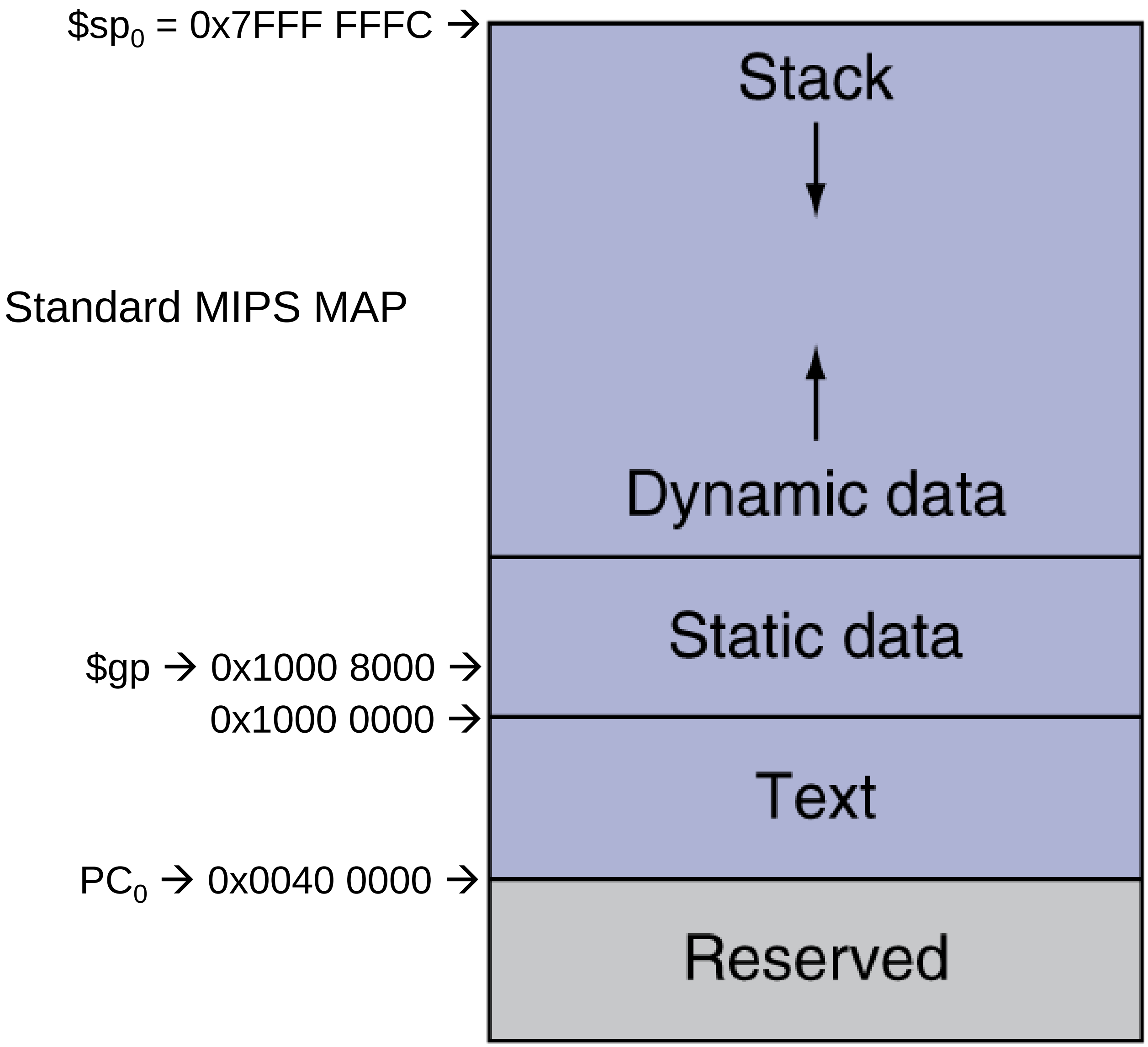
Extended ALU Control

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

opcode	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
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



$\$sp_0$ = 0x7FFF FFFC →

Standard MIPS MAP

$gp → 0x1000 8000 →

0x1000 0000 →

PC_0 → 0x0040 0000 →

MIPS Memory Layout/Map

$\$sp_0$ = 0x7FFF FFFC →

$\$sp_0$ = 0x7FFF EFFC →

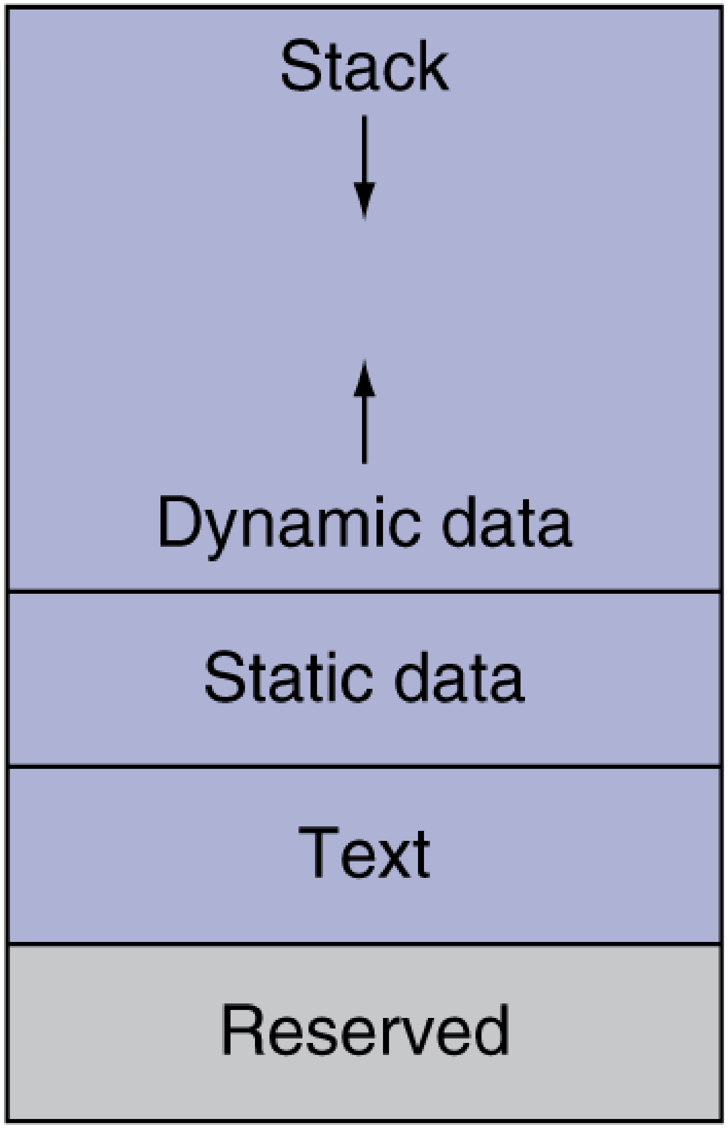
MIPS MAP in MARS

.data → 0x1001 0000 →

$gp → 0x1000 8000 →

0x1000 0000 →

PC_0 = 0x0040 0000 →



Stack

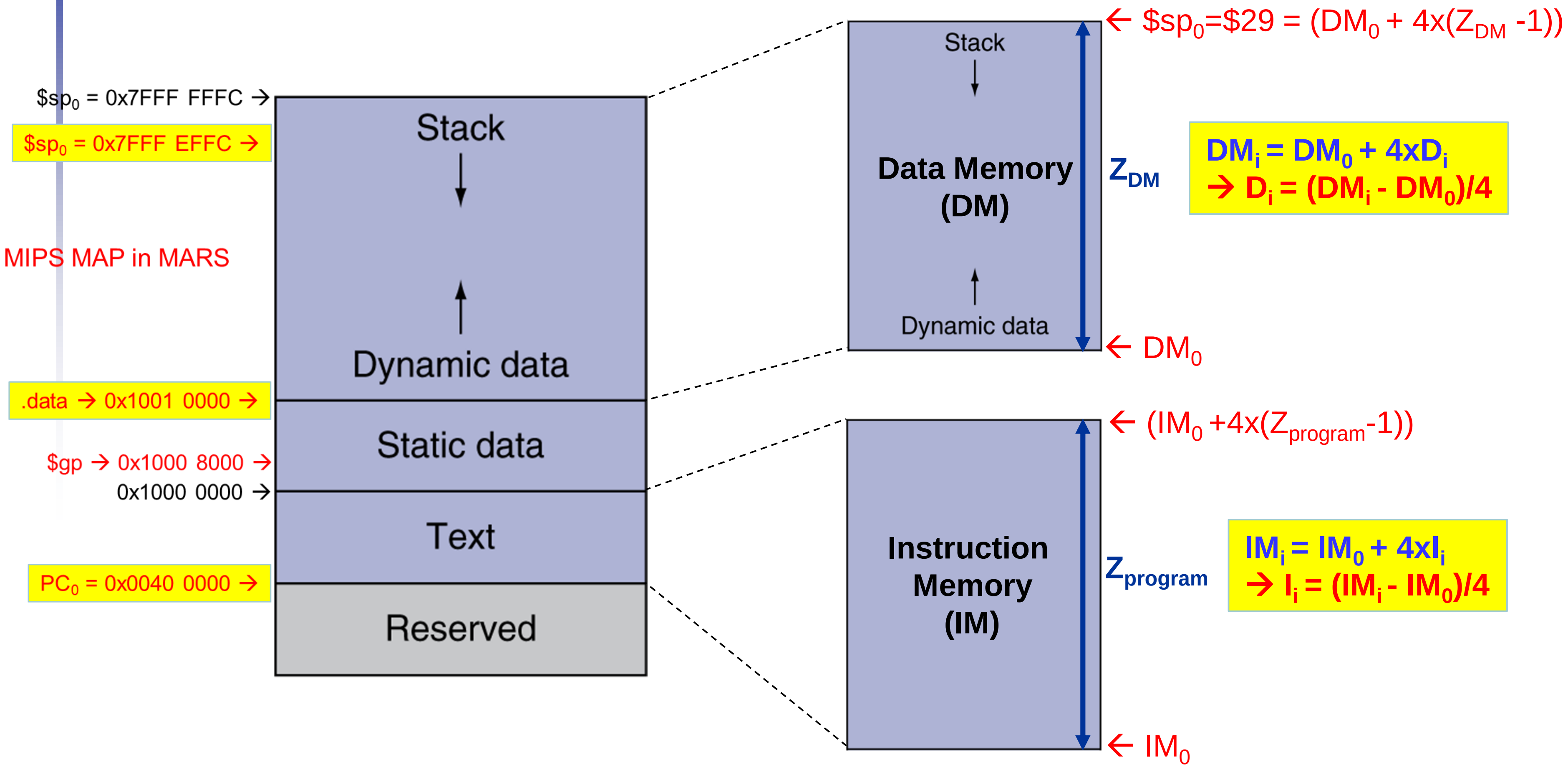
Dynamic data

Static data

Text

Reserved

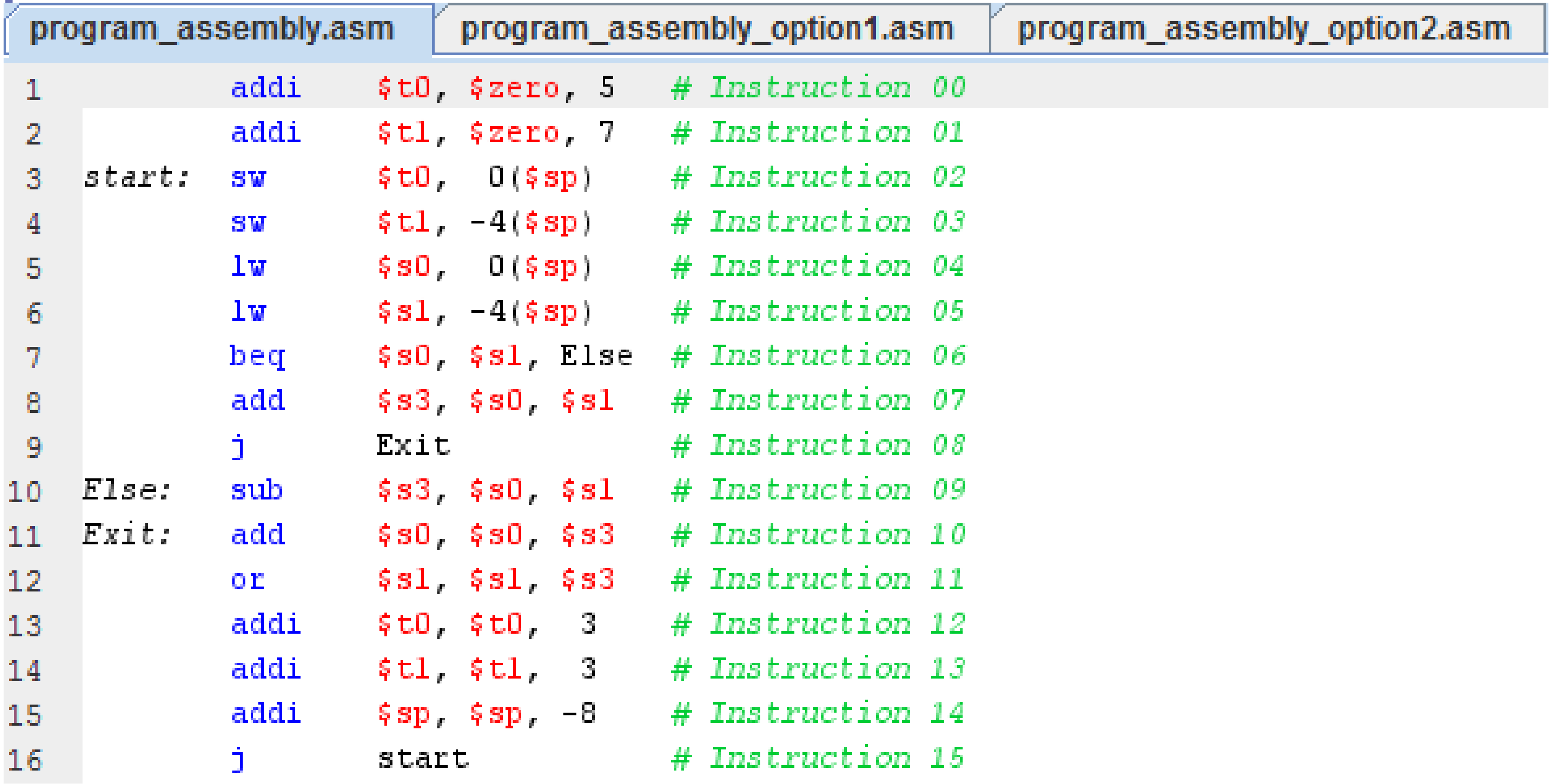
Mapping Your Data & Program



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**

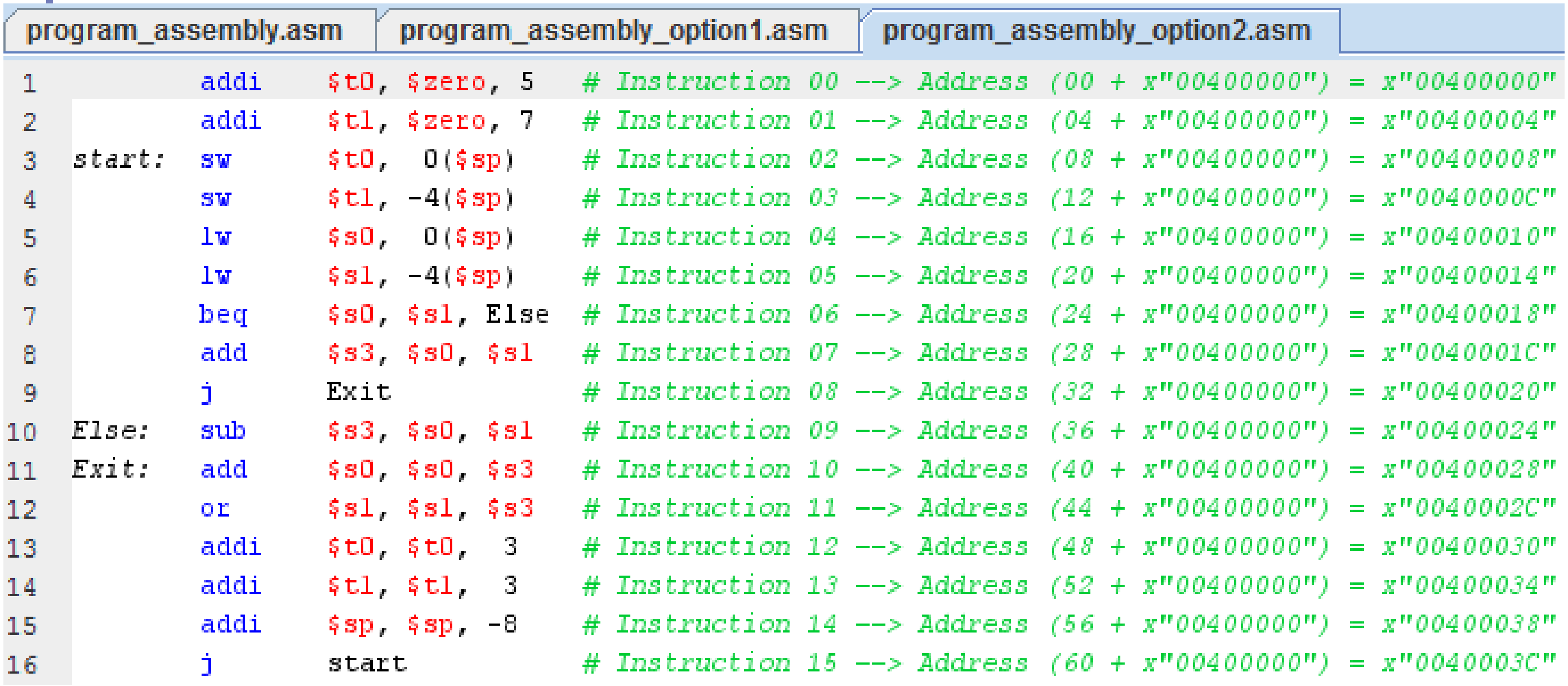
$Z_{program}$ ≡ Allocated size of instruction memory = Size of test program in 32-bit words (instructions)
IM_0 ≡ First address of instruction memory, could be anything, e.g. 0 or **0x0040 0000 = PC_0**

Mapping Your Program



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

Mapping Your Program - Option 2



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

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

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

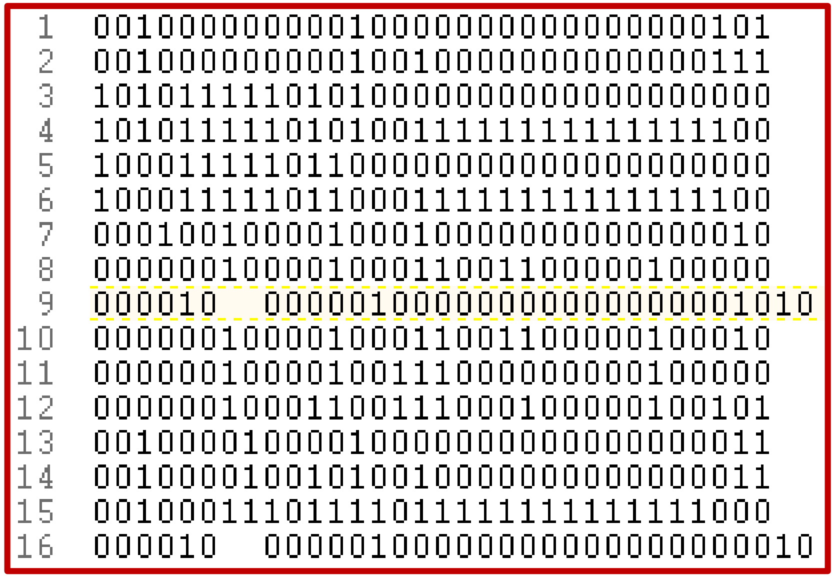
$DM_i = DM_0 + 4xD_i$
→ $D_i = (DM_i - DM_0)/4$

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

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

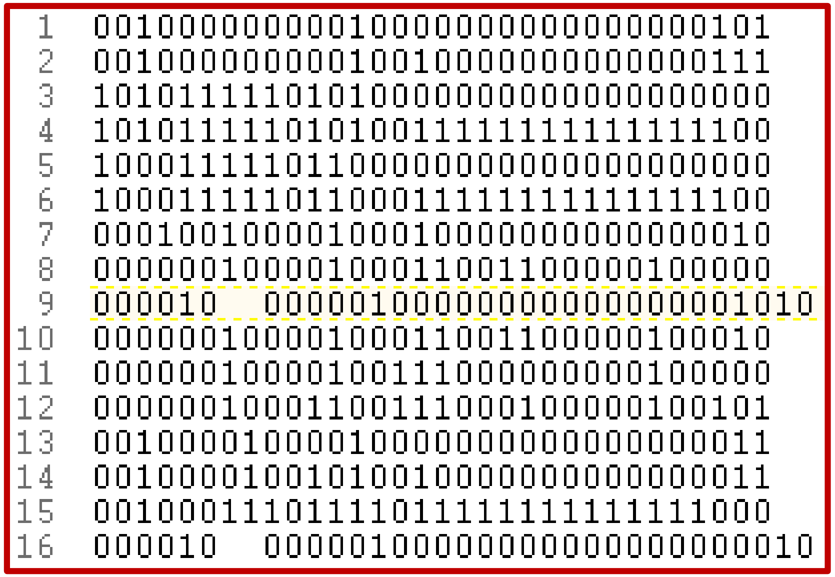
$DM_0 = A_0 = 0x1001\ 0000$
→ $\$sp_0 = A_0 + 4x(Z_{DM}\text{-}1)$

Mapping Your Program - Option 2



```
00100000000010000000000000000101
00100000000010010000000000000111
10101111101010000000000000000000
10101111101010011111111111111100
10001111101100000000000000000000
10001111101100011111111111111100
00010010000100010000000000000010
00000010000100011001100000100000
000010  00000100000000000000001010
00000010000100011001100000100010
00000010000100111000000000100000
00000010001100111000100000100101
00100001000010000000000000000011
00100001001010010000000000000011
00100011101111011111111111111000
000010  00000100000000000000000010
```

Manually-Assembled Program



```
00100000000010000000000000000101
00100000000010010000000000000111
10101111101010000000000000000000
10101111101010011111111111111100
10001111101100000000000000000000
10001111101100011111111111111100
00010010000100010000000000000010
00000010000100011001100000100000
000010  00000100000000000000001010
00000010000100011001100000100010
00000010000100111000000000100000
00000010001100111000100000100101
00100001000010000000000000000011
00100001001010010000000000000011
00100011101111011111111111111000
000010  00000100000000000000000010
```

MARS-Assembled Program

Same

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

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

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

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

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

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