

Report of Lab 2.8

Lalit Meena,2019CS50439

For this Lab 2 we are building microarchitecture for subset of arm instructions.

This lab involves putting together some stage 1 modules, to form a simple processor which can execute the following subset of ARM instructions with limited variants/features. {add, sub, cmp, mov, ldr, str, beq, bne, b}. Then in stage 3, we extend single cycle datapath(stage 2) to flexible and efficient clock period multicycle path.

In stage 4, we have tested our design exhaustively and reported waves/test-cases for most DP instructions. Then in stage 5, we are adding one more features to our implementation which supports shift operations on register and immediate operand variations. In stage 6, we are supporting more DT instructions features. It include byte and half word transfers (signed and unsigned), auto increment/decrement with option of pre/post indexing. Then in stage 7 we are including all variations of multiply instructions with mul_acc module

In stage 8 Included remaining instructions and features -bl, swi instructions, full predication

Zip folder name – L2.8_2019CS50439

Submission folder, L2.8_2019CS50439 contains-

1. **alu.vhd** – alu module design file
2. **data_mem.vhd** - data memory 128x32 module design file
3. **rfile.vhd** - design file for module register file 16x32
4. **others.vhd**-contains additional modules require for stage-2(Instruction Decoder, Condition Checker)-updated for full predication
5. **flags.vhd**- design file for module flags (**one line change**)
6. **multicycle.vhd**- SIMPLE multi CYCLE PROCESSOR
7. **shifter.vhd**- Shifter which support 4 shift types-LSL,LSR,ASR,ROR
8. **PM.vhd**- combination circuit path between the processor and memory
9. **Mul_acc.vhd**- module to perform multiply -accumulator computation
10. **package.vhd** -package to declare some types.-etc (**states added**)**run.do**

testbench files-

testbench_multicycle.vhd - testbench file for module simple multicycle processor

final TESTCASES EPWAVE PICS-ftestcase-1,2,3,4,5,6....

final TESTCASES files- ftestcasen.s, n is 1,2,.....

Report of Lab 2.8-lab 2 stage 8 description and test cases(also screenshots)

More details are in next pages –

stage 8 changes in main multicycle processor

there require appropriate adjustment, signals to workable main programme signals.

Number of states used inside main clocked process is states-

(fetch, read_AB, arithm, MSTR, MLDR, w2RF, READC, RSHIFT, WB2RF, MULAC, MW2RF, SWIST, RETRN, ELAST). Also used state signal in testbench to show at which state running currently.

At this stage our design have only 14 states ,here we HAVE added THREE new states –

swist-to branch swi related ISRs

retrn-to handle return type special cases -rte,ret

elast-to reach halted state when error produced ,program exit or access error related to priveledge mode

Format of new instructions.

bl #Signed_Offset

cond	10	11	Signed_Offset	
4	2	2	24	

swi.#0

cond	11	11	0	
4	2	2	24	

ret

cond	01	1	0	10000
4	2	1	20	5

rte

cond	01	1	0	10001
4	2	1	20	5

Memory organization-DATA MEMORY (128x32)

byte addresses going from 0 to 511, that is from 0x00000000 to 0x000001FF

SUPERVISOR -system AREA-0x00000000 to 0x000000FF (0 to 255)

INPUT PORT- 0x00000070

ISR of reset can be placed at 0x00000020 –8

E6000011-RTE

ISR for SWI can be placed at 0x00000040-10

```
mov r1,#0X00000070 @port address
loop : ldr r0,[r1]
tst r0,#0X00000100 @checking status bit
beq loop
and r0,r0,#0X000000FF @ascii code
@rte
```

E3A01070

E5910000

E3100C01

0AFFFFFC

E2400030

E6000011

Instructions to branch to these addresses need to be placed at –

RESET-0x00000000 and SWI-0x00000008

For this branch case the byte offsets are (0x20- 0x08) and (0x40 - 0x08 - 0x08), i.e., 0x18 and 0x30

word offset – 0x06 , 0xc

USER AREA-0x00000100 to 0x000001FF (256 to 511)

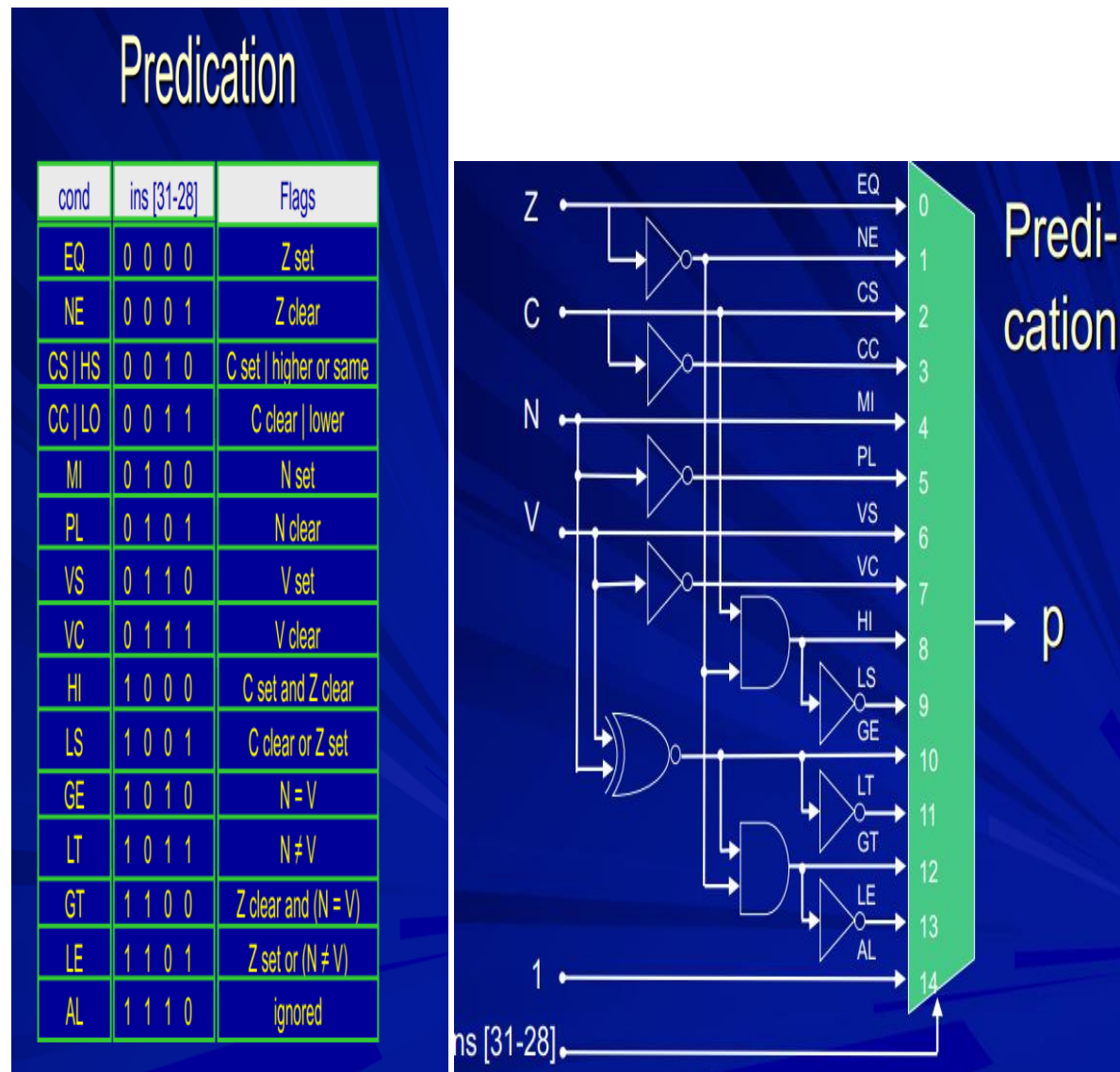
User program starts from 0x00000100

Statck for user memory usage

R13-sp - 0x000001FF

Full predication logic in checker module-

Checker module was previously tested in others.vhd.



Flags module is more tested here as also previously tested at each stage.

How DP instructions affect Flags C, V, Z, N

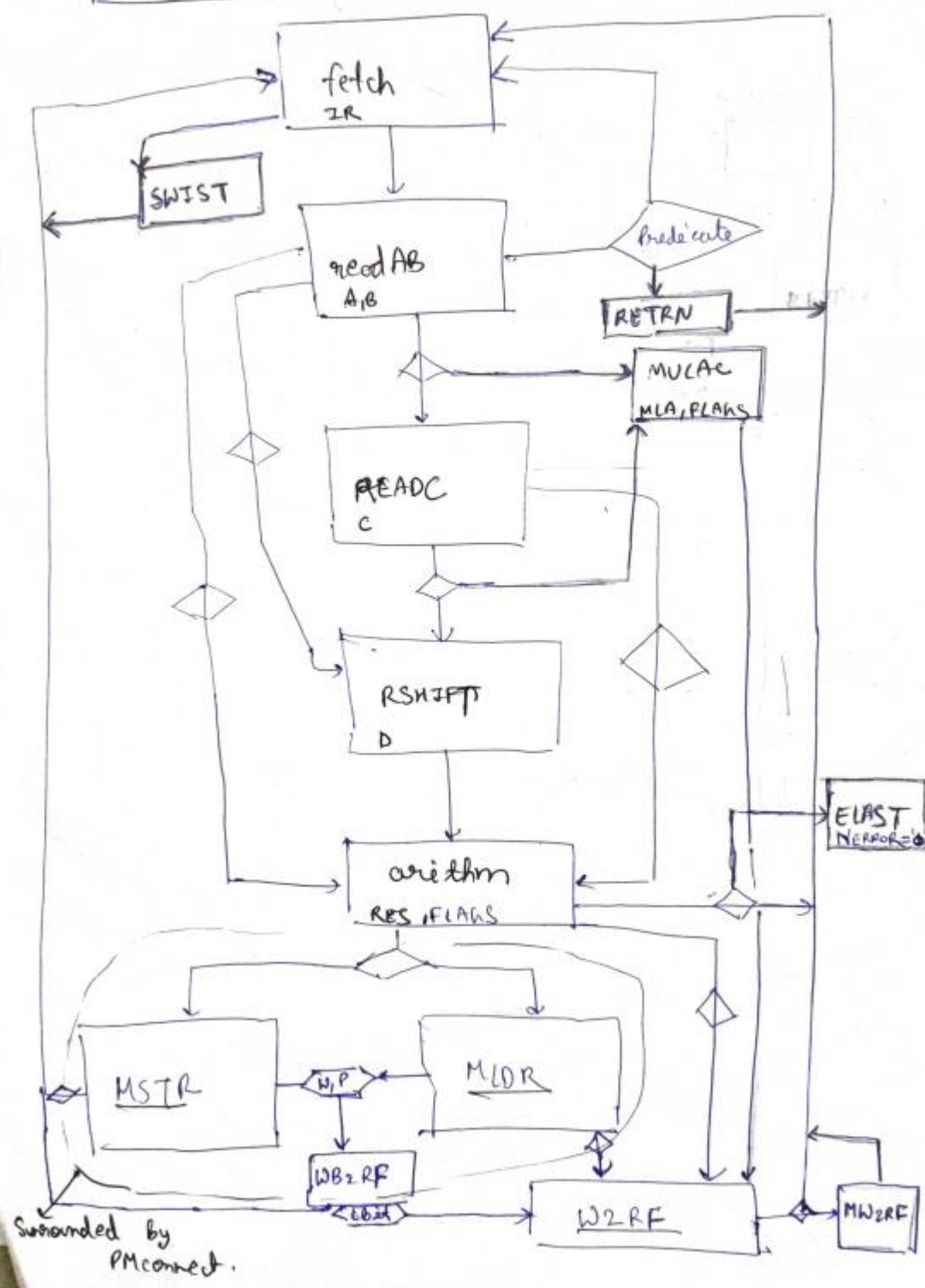
Instructions	Effect on Flags		
	if S-bit = 0	if S-bit = 1 and no shift/rotate	if S-bit = 1 and shift/rotate is there
add, sub, rsb, adc, sbc, rsc	No flags are affected	All 4 flags are affected, ALU carry is used	
cmp, cmn	All 4 flags are affected, ALU carry is used		
and, orr, xor, bic, mov, mvn	No flags are affected	Only Z and N are affected	C, Z and N are affected shift/rotate carry is used
tst, teq	Only Z and N are affected		

Flags are setup exhaustively to handle shifter,multiply,carry etc all type variations.

Page-8.2

(16 Notes)

ASM CHART



Testcases

simulated with help of testbench multicycle.vhd-

state change like flags and registers at end of program or step can be checked by signal like A,B,C,D,FLAGS, IR,PC,RES,state,minstype ,MWetc.

Input-port testing

Here main things to look – mode signal and input-testbench related parameters(inpt-testbench setting signal to set input port ,indata(status+byte data).As in testbench combination of two are setting input port location in data memory.

After changing the input data, the status signal (9th bit) should become 1 for several clock cycles and then become 0 till the next data change

To handle it we **multiplexed signals** from **PMConnect** and **these testbench** to go in memory.

```
entity processor is
  port (
    clock, reset : in std_logic;
    INPT : in std_logic;
    INDATA : IN WORD
  );
end processor;
```

Correctness can be justified by values of registers read after instructions,etcAlso added extra mov instructions to read and show value of B,RES OR 2nd operand.

For errors check d-state(elast) .

We can observe number of cycles etc,also used state signal to show which state currently

Below are related testcases-which covers cases told to implement-

Read from input-storing memory

From User mode try system area access

BL and ret type

Full predication capability

Flags -sbit

Each testcase was little explained and also ownself explanatory.

Take help of **run time limit** on simulator for taking required PC increments.For clock related testbench signals .

TESTCASES SEPARATE EPWAVE PICS and assembly files CAN also BE FOUND IN SUBMITTED FOLDER.

Ftestcase1

Compare ftestcase1.jpg with testcase3.jpg, only initial 2 reset related instructions are extra.

The screenshot shows a debugger interface with two main windows: **RegistersView** and **CodeView**.

RegistersView: The **General Purpose** tab is selected. It shows a list of registers (R0 to R15) and the CPSR Register. The values are as follows:

Register	Value
R0	: 00000002
R1	: 00000003
R2	: 00000005
R3	: 00000005
R4	: 00000000
R5	: 00000000
R6	: 00000000
R7	: 00000000
R8	: 00000000
R9	: 00000000
R10 (s1)	: 00000000
R11 (fp)	: 00000000
R12 (ip)	: 00000000
R13 (sp)	: 00011400
R14 (lr)	: 00000000
R15 (pc)	: 00011400

Below the registers, the **CPSR Register** status is shown:

- Negative (N) : 0
- Zero (Z) : 1
- Carry (C) : 1
- Overflow (V) : 0

CodeView: The **CodeView** window shows the assembly code for **ftestcase1.o**. The code is as follows:

```

@normal test
.text
00001000:E3A00002  mov r0,#2
00001004:E3A01003  mov r1,#3
00001008:E0812000  add r2,r1,r0
0000100C:E3520005  cmp r2,#5
00001010:01A03002  moveq r3,r2
.end
  
```

The **OutputView** and **WatchView** tabs are also visible at the bottom, with the **Console** tab selected.

Ftestcase2

VERIFY 0x32 -2 ascii code input byte by seeing last instruction A register

The screenshot shows the **CodeView** window for **ftestcase2.o**. The code is as follows:

```

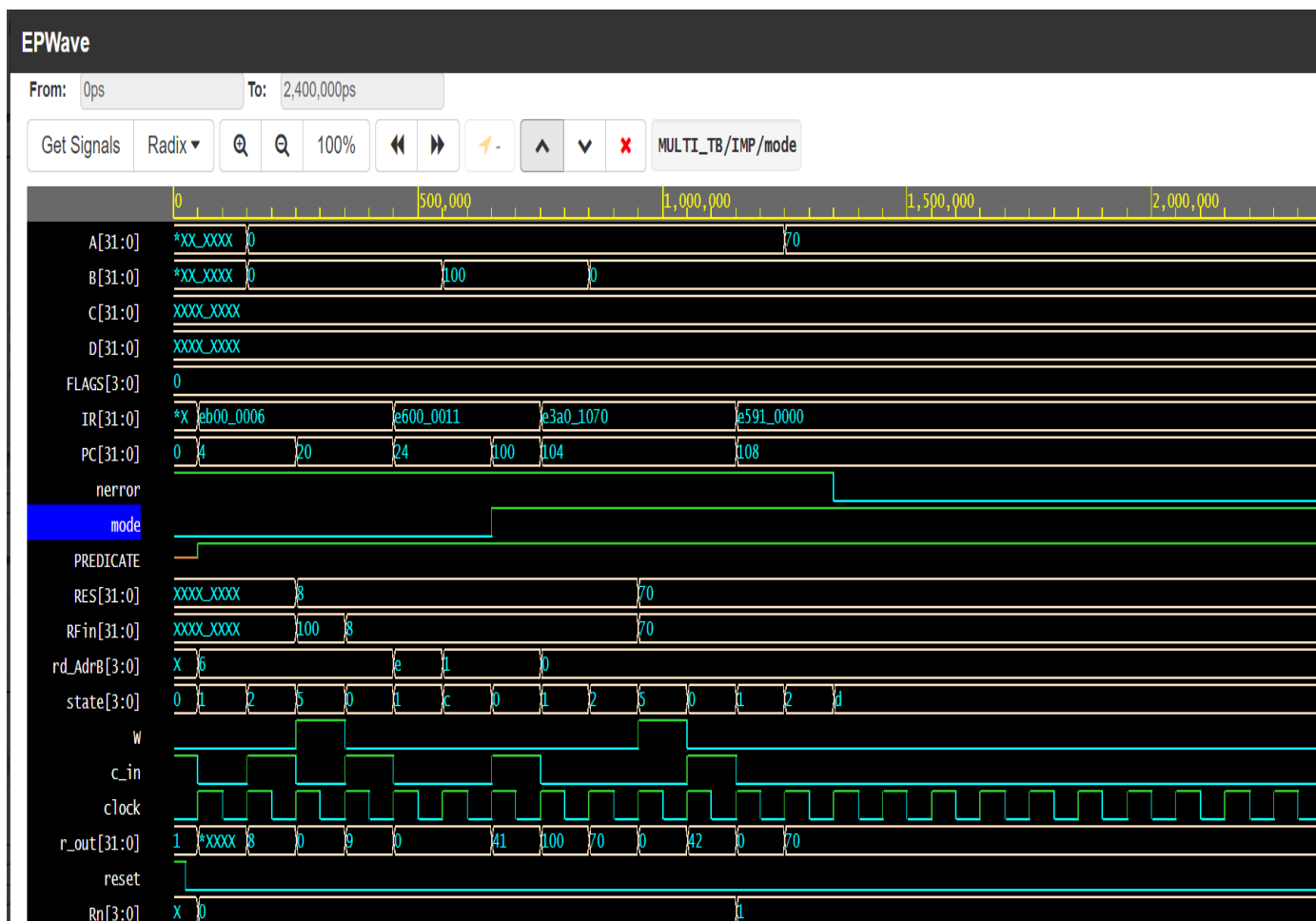
@test to read from input port and storing in memory (satck partition-last)
@later performing operation to verify correctness
.text
00001000:EF000000  swi #0
00001004:E44D0004  strb r0,[r13] ,#-4 @stack
00001008:E2800002  add r0,r0,#2
.end...
  
```

Ftestcase3

When trying to access supervisor partitioned while in user mode-

Program halt by going to error state(d-elast ,nerror = '1')

```
CodeView
fptestcase3.o
    @test to try read inport port in user mode
    @but at return result in exit/elast state
    .text
00001000:E3A01070  mov r1,#0X00000070 @port address
00001004:E5910000  ldr r0,[r1]
00001008:E1A02000  mov r2,r0
    .end...
```



Fptestcase4-

BL and ret type implemenatation with simple 2's power function-8 result

users are freely able to execute nested calls etc.but ownself responsible for appropriate steps like saving registers, previous lr on stack partition-r13

The screenshot shows a debugger interface with a register list on the left and an assembly view on the right. The register list shows R0 through R15 with their current values. The assembly view shows the execution of a program named 'ftestcase4.o'. The code includes comments about BL and RET instructions, a function 'fun' that calculates 2's power, and a return statement. The current instruction being executed is 'mov r0, #3' at address 00001000.

Register	Value
R0	:00000008
R1	:00000001
R2	:00000000
R3	:00000000
R4	:00000000
R5	:00000000
R6	:00000000
R7	:00000000
R8	:00000000
R9	:00000000
R10 (s1)	:00000000
R11 (fp)	:00000000
R12 (ip)	:00000000
R13 (sp)	:00011400
R14 (lr)	:00001008
R15 (pc)	:00011400

```
@simple BL and RET instructions
@ users are freely able to execute nested calls etc.
@but ownself responsible for appropriate steps like saving registers, previous lr on stack partitio
.text
00001000:E3A00003  mov r0,#3
00001004:EB000000    bl fun
00001008:E1A01000  mov r1,r0

@function to give 2's power r0
0000100C:E3A01001  fun: mov r1,#1
00001010:E1A00011  mov r0,r1,LSL r0
@ret
.end...
```

Ftestcase5

test-full predication on complex shifter based instructions,here C,Z get set

The screenshot displays a debugger window with the following components:

- RegistersView:** Shows the state of 16 registers (R0-R15) and the CPSR register. R0-R9 are in hexadecimal, while R10-R15 and CPSR are in decimal. R15 (PC) is highlighted in red.
- CodeView:** Shows the assembly code for the test case, with the first instruction highlighted in blue.
- OutputView/WatchView:** Currently empty.
- Console:** Shows the standard input/output streams (stdin/stdout/stderr).

Register	Value
R0	:00000002
R1	:00000100
R2	:00000000
R3	:00000002
R4	:00000000
R5	:00000002
R6	:00000000
R7	:00000000
R8	:00000000
R9	:00000000
R10 (s1)	:00000000
R11 (fp)	:00000000
R12 (ip)	:00000000
R13 (sp)	:00011400
R14 (lr)	:00000000
R15 (pc)	:00011400

CPSR Register	Value
Negative (N)	:0
Zero (Z)	:1
Carry (C)	:1
Overflow (V)	:0
IRQ Disable	:1
FIQ Disable	:1
Thumb (T)	:0
CPU Mode	:System


```
@test-full predication on complex shifter based instructions,here C,Z get set
.text
00001000:E3A00002  mov r0,#2
00001004:E3A01C01  mov r1,#256
00001008:E0112120  ands r2,r1,r0,LSR #2
0000100C:E1A04002  moval r4,r2
00001010:03A03000  moveq r3,#0
00001014:23A03002  movhs r3,#2
00001018:33A03003  movlo r3,#3
0000101C:E1A05003  mov r5,r3
.end...
```

Ftestcase6

test to cover possible variations full predication and sbit

The screenshot displays a debugger interface with three main panels. The left panel, titled 'RegistersView', shows a list of registers R0 through R15 and the CPSR register. R0-R9 are in 'General Purpose' mode and show 00000000. R10 (s1) is 00000000. R11 (fp) is 00000000. R12 (ip) is 00000000. R13 (sp) is 00011400. R14 (lr) is 00000000. R15 (pc) is 00011400. The CPSR register shows Negative (N) : 0, Zero (Z) : 1, Carry (C) : 0, and Overflow (V) : 0. The middle panel, titled 'CodeView', shows assembly code for 'ftestcase6.o'. The code starts with a comment '@test to cover possible variations full predication and sbit', followed by '.text', then '00001000:E3B00000 movs r0,#0', '00001004:13A01001 movne r1,#1', '00001008:B3A0100B movlt r1,#11', '0000100C:C3A0100C movgt r1,#12', '00001010:53A01005 movpl r1,#5', '00001014:E1A02001 mov r2,r1', and ends with '.end...'. The right panel, titled 'OutputView', shows 'Console' output with 'stdin/stdout/stderr' selected.

RegistersView

General Purpose Floating Point

Hexadecimal

Unsigned Decimal

Signed Decimal

R0 : 00000000

R1 : 00000005

R2 : 00000005

R3 : 00000000

R4 : 00000000

R5 : 00000000

R6 : 00000000

R7 : 00000000

R8 : 00000000

R9 : 00000000

R10 (s1) : 00000000

R11 (fp) : 00000000

R12 (ip) : 00000000

R13 (sp) : 00011400

R14 (lr) : 00000000

R15 (pc) : 00011400

CPSR Register

Negative (N) : 0

Zero (Z) : 1

Carry (C) : 0

Overflow (V) : 0

CodeView

ftestcase6.o

@test to cover possible variations full predication and sbit

.text

00001000:E3B00000 movs r0,#0

00001004:13A01001 movne r1,#1

00001008:B3A0100B movlt r1,#11

0000100C:C3A0100C movgt r1,#12

00001010:53A01005 movpl r1,#5

00001014:E1A02001 mov r2,r1

.end...

OutputView WatchView

Console stdin/stdout/stderr

Ftestcase7

Flags and predication related variations

The screenshot displays a debugger interface with three main panels. The left panel, titled 'RegistersView', shows the state of 16 general-purpose registers (R0-R15) and the CPSR register. The middle panel, titled 'CodeView', shows the assembly code for a file named 'fptestcase7.o'. The bottom panel, titled 'OutputView', shows the console output.

RegistersView:

Register	Value
R0	:00049000
R1	:80000002
R2	:00000002
R3	:00092002
R4	:00000000
R5	:00092002
R6	:00000000
R7	:00000000
R8	:00000000
R9	:00000000
R10 (s1)	:00000000
R11 (fp)	:00000000
R12 (ip)	:00000000
R13 (sp)	:00011400
R14 (lr)	:00000000
R15 (pc)	:00011400

CPSR Register:

Flag	Value
Negative (N)	:0
Zero (Z)	:0
Carry (C)	:0
Overflow (V)	:0
IRQ Disable	:1
FIQ Disable	:1

CodeView:

```
.text
00001000:E3A00A49  mov r0,#0X00049000
00001004:E3A0110A  mov r1,#0X80000002
00001008:E3A02002  mov r2,#2
0000100C:E0332091  mlas r3,r1,r0,r2
00001010:11A05003  movne r5,r3
.end...
```

OutputView:

Console: stdin/stdout/stderr

Ftestcase8

Flags and predication related variations

The screenshot displays a debugger window with two main panes. The left pane shows the state of CPU registers and the CPSR register. The right pane shows the assembly code for a file named `fptestcase8.o`.

Register Values:

Register	Value
R0	00000007
R1	00000004
R2	00000004
R3	00000000
R4	00000000
R5	00000000
R6	00000000
R7	00000000
R8	00000000
R9	00000000
R10 (s1)	00000000
R11 (fp)	00000000
R12 (ip)	00000000
R13 (sp)	00011400
R14 (lr)	00000000
R15 (pc)	00011400

CPSR Register:

Flag	Value
Negative (N)	0
Zero (Z)	1
Carry (C)	0
Overflow (V)	0
IRQ Disable	1
FIQ Disable	1

Assembly Code:

```
.text
00001000:E3A00007  mov r0,#7
00001004:E3A01004  mov r1,#4
00001008:E0112000  ands r2,r1,r0
0000100C:00623001  rsbeq r3,r2,r1
00001010:E3130000  tst r3,#0
.end...
```

Thank you!