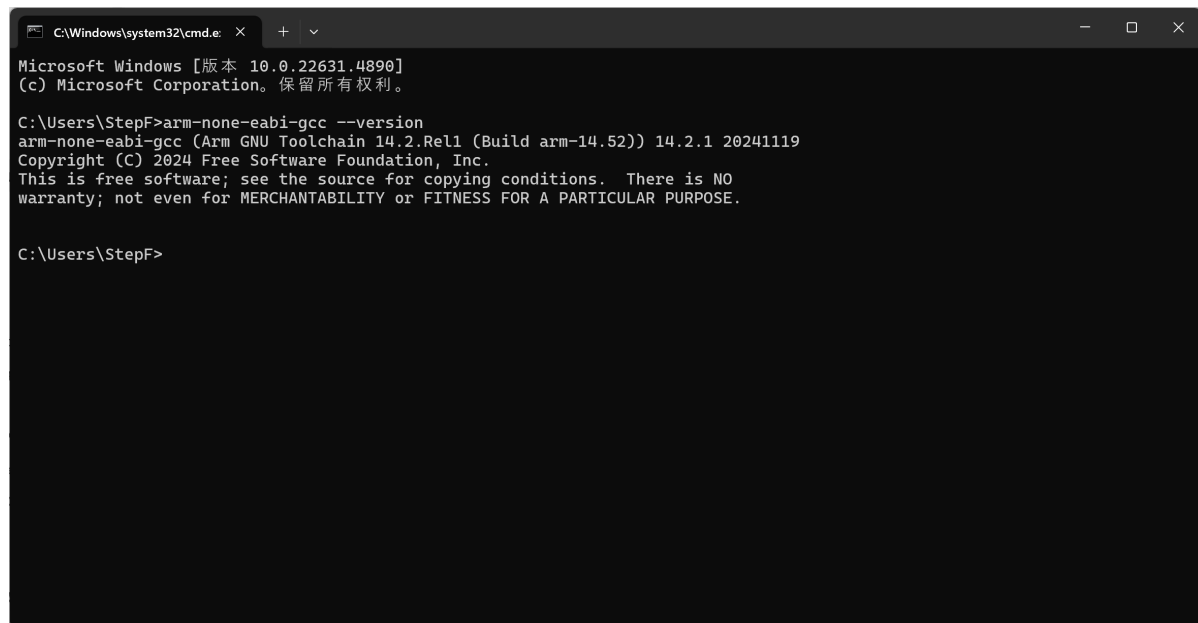
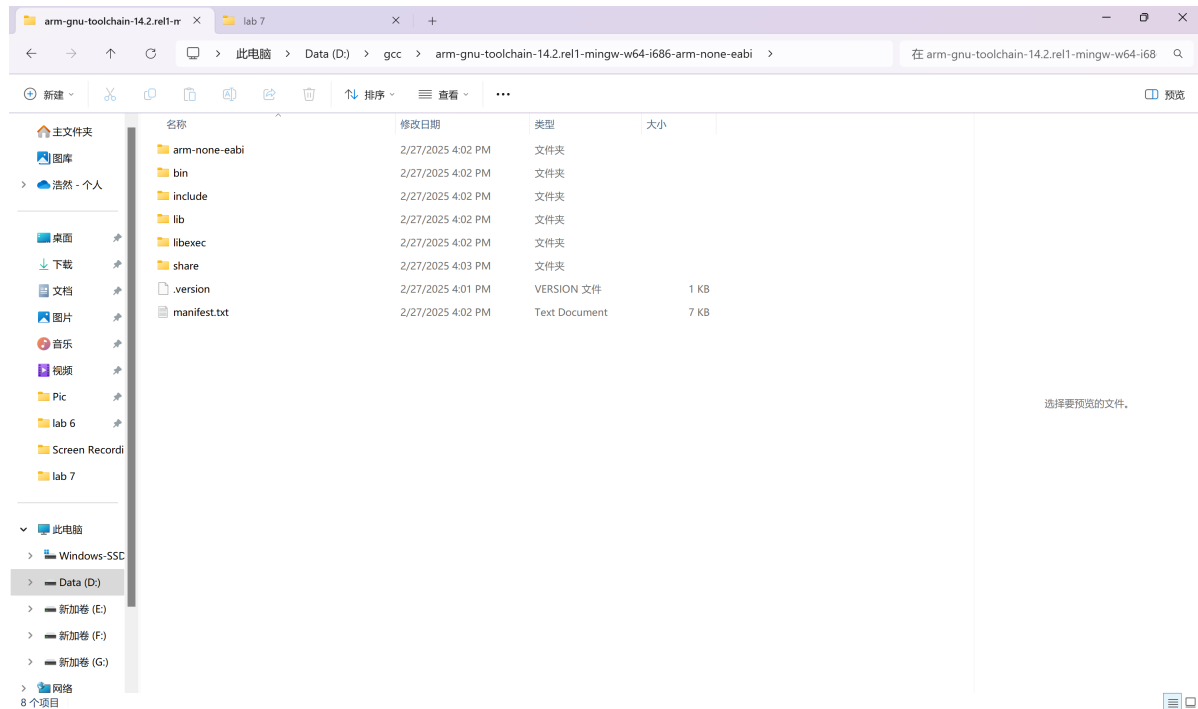


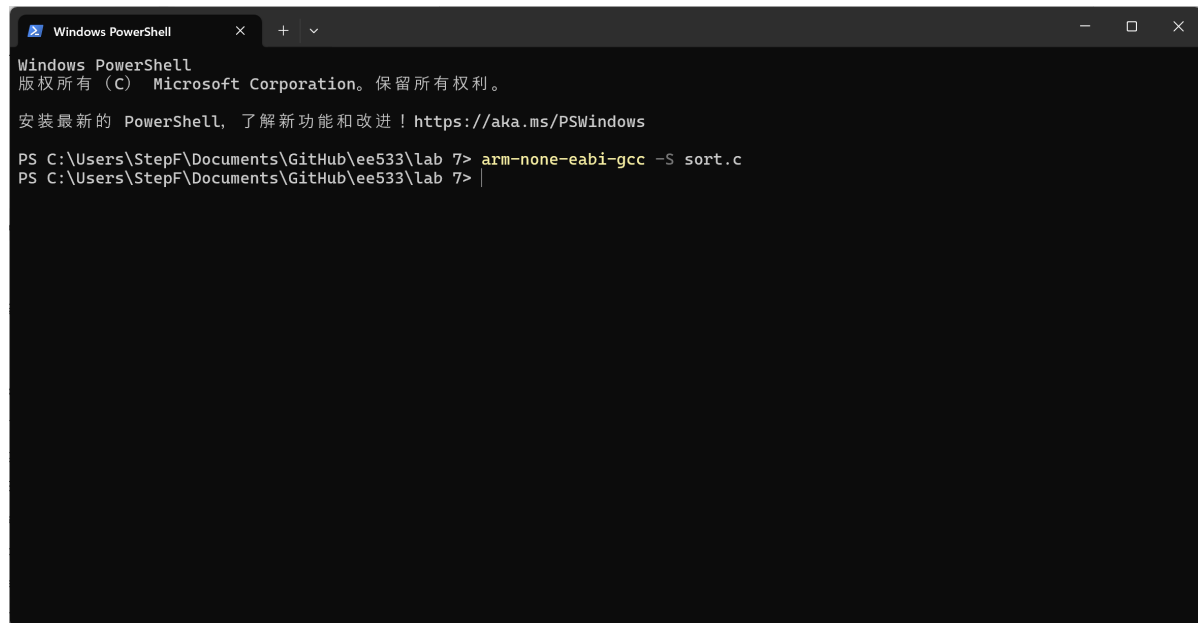
EE533_Lab7_Report

1. Find what Instruction We Need to Implement

1.1 ARM C/C++ Cross-compiler Installation



1.2 Assembly Code for Bubble Sort



```
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows

PS C:\Users\StepF\Documents\GitHub\ee533\lab 7> arm-none-eabi-gcc -S sort.c
PS C:\Users\StepF\Documents\GitHub\ee533\lab 7> |
```

- sort.c

```
int main() {
    int array[10] = {323, 123, -455, 2, 98, 125, 10, 65, -56, 0};
    int i, j, swap;
    for (i = 0 ; i < 10; i++) {
        for (j = i+1 ; j < 10 ; j++) {
            if (array[j] < array[i]) {
                swap = array[j];
                array[j] = array[i];
                array[i] = swap;
            }
        }
    }
}
```

- sort.s

```
.cpu arm7tdmi
.arch armv4t
.fpu softvfp
.eabi_attribute 20, 1
.eabi_attribute 21, 1
.eabi_attribute 23, 3
.eabi_attribute 24, 1
.eabi_attribute 25, 1
.eabi_attribute 26, 1
.eabi_attribute 30, 6
.eabi_attribute 34, 0
.eabi_attribute 18, 4
.file "sort.c"
.text
.section .rodata
.align 2
.LC0:
```

```

.word    323
.word    123
.word    -455
.word    2
.word    98
.word    125
.word    10
.word    65
.word    -56
.word    0
.text
.align   2
.global  main
.syntax  unified
.arm
.type    main, %function
main:
    @ Function supports interworking.
    @ args = 0, pretend = 0, frame = 56
    @ frame_needed = 1, uses_anonymous_args = 0
    push    {fp, lr}
    add fp, sp, #4
    sub sp, sp, #56
    ldr r3, .L8
    sub ip, fp, #56
    mov lr, r3
    ldmia   lr!, {r0, r1, r2, r3}
    stmia   ip!, {r0, r1, r2, r3}
    ldmia   lr!, {r0, r1, r2, r3}
    stmia   ip!, {r0, r1, r2, r3}
    ldm lr, {r0, r1}
    stm ip, {r0, r1}
    mov r3, #0
    str r3, [fp, #-8]
    b      .L2
.L6:
    ldr r3, [fp, #-8]
    add r3, r3, #1
    str r3, [fp, #-12]
    b      .L3
.L5:
    ldr r3, [fp, #-12]
    lsl r3, r3, #2
    sub r3, r3, #4
    add r3, r3, fp
    ldr r2, [r3, #-52]
    ldr r3, [fp, #-8]
    lsl r3, r3, #2
    sub r3, r3, #4
    add r3, r3, fp
    ldr r3, [r3, #-52]
    cmp r2, r3
    bge .L4
    ldr r3, [fp, #-12]
    lsl r3, r3, #2
    sub r3, r3, #4

```

```

    add r3, r3, fp
    ldr r3, [r3, #-52]
    str r3, [fp, #-16]
    ldr r3, [fp, #-8]
    lsl r3, r3, #2
    sub r3, r3, #4
    add r3, r3, fp
    ldr r2, [r3, #-52]
    ldr r3, [fp, #-12]
    lsl r3, r3, #2
    sub r3, r3, #4
    add r3, r3, fp
    str r2, [r3, #-52]
    ldr r3, [fp, #-8]
    lsl r3, r3, #2
    sub r3, r3, #4
    add r3, r3, fp
    ldr r2, [fp, #-16]
    str r2, [r3, #-52]
.L4:
    ldr r3, [fp, #-12]
    add r3, r3, #1
    str r3, [fp, #-12]
.L3:
    ldr r3, [fp, #-12]
    cmp r3, #9
    ble .L5
    ldr r3, [fp, #-8]
    add r3, r3, #1
    str r3, [fp, #-8]
.L2:
    ldr r3, [fp, #-8]
    cmp r3, #9
    ble .L6
    mov r3, #0
    mov r0, r3
    sub sp, fp, #4
    @ sp needed
    pop {fp, lr}
    bx lr
.L9:
    .align 2
.L8:
    .word .LC0
    .size main, .-main
    .ident "GCC: (Arm GNU Toolchain 14.2.Rel1 (Build arm-14.52)) 14.2.1
20241119"

```

- Included Instructions' Description and Reference

Instruction	Explanation	Reference
add	Add	https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set/general-data-processing-instructions/add--adc--sub--sbc--and-rsb
sub	Subtract	https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set/general-data-processing-instructions/add--adc--sub--sbc--and-rsb
mov	Move	https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set/general-data-processing-instructions/mov-and-mvn
ldm	Load Multiple registers, increment after	https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set/memory-access-instructions/ldm-and-stm
ldmia	LDMIA is a synonym for LDM	https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set/memory-access-instructions/ldm-and-stm
stm	Store Multiple registers, increment after	https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set/memory-access-instructions/ldm-and-stm
stmia	STMIA is a synonym for STM	https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set/memory-access-instructions/ldm-and-stm
ldr	Load Register with word	https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set/memory-access-instructions/ldr-and-str--register-offset
str	Store Register word	https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set/memory-access-instructions/ldr-and-str--register-offset
lsl	Logical Shift Left	https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set/general-data-processing-instructions/asr--lsl--lsr--ror--and-rrx?lang=en
cmp	Compare	https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set/general-data-processing-instructions/cmp-and-cmn
push	Push registers onto stack	https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set/memory-access-instructions/push-and-pop
pop	Pop registers from stack	https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set/memory-access-instructions/push-and-pop

Instruction	Explanation	Reference
b	Branch	https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set/branch-and-control-instructions/b--bl--bx--and-blx
bge	Branch if greater than or equal	https://developer.arm.com/documentation/100076/0100/A64-Instruction-Set-Reference/A64-General-Instructions/B-cond/Condition-code-suffixes-and-related-flags
ble	Branch if less or equal	https://developer.arm.com/documentation/100076/0100/A64-Instruction-Set-Reference/A64-General-Instructions/B-cond/Condition-code-suffixes-and-related-flags
bx	Indirect branch	https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set/branch-and-control-instructions/b--bl--bx--and-blx

1.3 List of instruction needs to be implemented

1.3.1 Instruction Format Definition

- Instruction Format definition

OP_CODE	Rs	Rt	Rd	Shift	Funct
I_MEM[31:26]	I_MEM[25:21]	I_MEM[20:16]	I_MEM[15:11]	I_MEM[10:6]	I_MEM[5:0]

- OP_Code look up table

Instruction	OP_Code	Description
add	000000	Add
sub	000001	Subtract
mov	000010	Move
ldr	000011	Load Register with word
str	000100	Store Register word
ldm	000101	Load Multiple registers, increment after
stm	000110	Store Multiple registers, increment after
ldmia	000111	LDMIA is a synonym for LDM
stmia	001000	STMIA is a synonym for STM
lsl	001001	Logical Shift Left
cmp	001010	Compare

Instruction	OP_Code	Description
push	001011	Push registers onto stack
pop	001100	Pop registers from stack
b	001101	Branch
bge	001110	Branch if greater than or equal
ble	001111	Branch if less or equal
bx	010000	Indirect branches

1.3.2 Special Registers in ARM ISA

Register Name	Actual Register Location	Description
FP	R11	Frame Pointer
IP	R12	Intra Procedural Call
SP	R13	Stack Pointer
LR	R14	Link Register

2. Pipelined Processor on NetFPGA

2.1 Instruction Generated

```
.data
array: .dword 323, 123, -455, 2, 98, 125, 10, 65, -56, 0
N:     .dword 10

.text
.global _start
_start:
    ldr r4, =array      @ r4 = base address of array
    ldr r5, =N           @ r5 = address of N
    ldr r5, [r5]         @ r5 = N (size of array)
    sub r5, r5, #1       @ r5 = N-1 (outer loop limit)

outer_loop:
    mov r6, #0           @ i = 0

inner_loop:
    sub r7, r6, r5       @ if i >= N-1, exit inner loop
    bge outer_continue

    @ Load array[i] and array[i+1]
    mov r8, r6, LSL #2   @ r8 = i * 4 (word offset)
    add r9, r4, r8       @ r9 = address of array[i]
    ldr r10, [r9]        @ r10 = array[i]
    ldr r11, [r9, #4]    @ r11 = array[i+1]
```

```

    sub r12, r10, r11    @ r12 = array[i] - array[i+1]
    ble no_swap          @ If array[i] <= array[i+1], no swap

    str r11, [r9]         @ array[i] = array[i+1]
    str r10, [r9, #4]     @ array[i+1] = array[i]

no_swap:
    add r6, r6, #1        @ i++
    b inner_loop

outer_continue:
    sub r5, r5, #1        @ Reduce loop limit (N-1, N-2, ...)
    bgt outer_loop        @ If still positive, loop again

end:
    b end                @ Infinite loop (halt)

```

- Assembly Code

```

#0  ldr r4, =array
#1  ldr r5, =N
#2  ldr r5, [r5]
#3  sub r5, r5, #1        @ outer_continue
#4  mov r6, #0           @ outer_loop
#5  mov r8, r6
#6  lsl r8, r8, #2
#7  add r9, r4, r8
#8  ldr r10, [r9]
#9  ldr r11, [r9, #8]
#10 sub r12, r10, r11
#11 ble no_swap
#12 str r11, [r9]
#13 str r10, [r9, #8]
#14 add r6, r6, #1        @ no_swap
#15 sub r7, r6, r5        @ inner_loop
#16 bge outer_continue
#17 b inner_loop
#18 b end                @ end

```

2.2 Instruction Format

2.2.1 Pesudo Instruction

```

lw r0, array_addr
lw r1, array_size
subi r1, r1, #1
mov r2, #0
outer_loop:
    beq r2, r1, end
    addi r3, r2, #1
inner_loop:
    bgt r3, r1, next_out
    lw r4, r0(r2)
    lw r5, r0(r3)

```



```

blt r4, r5, no_swap
sw r4, r0(r3)
sw r5, r0(r2)
no_swap:
addi r3, r3, #1
j inner_loop
next_out:
addi r2, r2, #1
j outer_loop
end:
j end

```

2.2.2 Real Instruction

- With manually introduced NOOP, we can avoid data dependency problem and branch flush problem.

```

movi r1, #9
outer_loop:
noop
noop
beq r2, r1, end
noop
addi r3, r2, #1
inner_loop:
noop
noop
bgt r3, r1, next_out
noop
lw r4, r2(#0)
lw r5, r3(#0)
noop
noop
blt r4, r5, no_swap
noop
sw r4, r3(#0)
sw r5, r2(#0)
no_swap:
addi r3, r3, #1
j inner_loop
noop
next_out:
addi r2, r2, #1
j outer_loop
noop
end:
j end

```

- Instruction OP Code

Instr	OP Code [31:26]
noop	000000

Instr	OP Code [31:26]
addi	000001
movi	000010
lw	000011
sw	000100
beq	000101
bgt	000110
blt	000111
j	001000

Addr	Label	Instr	OP Code [31:26]	Rs [25:21]	Rt [20:16]	Offset [15:0]
0		movi r1, #9	000010	5'd0	5'd1	16'd9
1	outer_loop	noop	000000	5'd0	5'd0	16'd0
2		noop	000000	5'd0	5'd0	16'd0
3		beq r2, r1, end	000101	5'd1	5'd2	16'd24
4		noop	000000	5'd0	5'd0	16'd0
5		addi r3, r2, #1	000001	5'd2	5'd3	16'd1
6	inner_loop	noop	000000	5'd0	5'd0	16'd0
7		noop	000000	5'd0	5'd0	16'd0
8		bgt r3, r1, next_out	000110	5'd1	5'd3	16'd21
9		noop	000000	5'd0	5'd0	16'd0
10		lw r4, r2(#0)	000011	5'd2	5'd4	16'd0
11		lw r5, r3(#0)	000011	5'd3	5'd5	16'd0
12		noop	000000	5'd0	5'd0	16'd0
13		noop	000000	5'd0	5'd0	16'd0
14		blt r4, r5, no_swap	000111	5'd5	5'd4	16'd18
15		noop	000000	5'd0	5'd0	16'd0

Addr	Label	Instr	OP Code [31:26]	Rs [25:21]	Rt [20:16]	Offset [15:0]
16		sw r4, r3(#0)	000100	5'd3	5'd4	16'd0
17		sw r5, r2(#0)	000100	5'd2	5'd5	16'd0
18	no_swap	addi r3, r3, #1	000001	5'd3	5'd3	16'd1
19		j inner_loop	001000	5'd0	5'd0	16'd6
20		noop	000000	5'd0	5'd0	16'd0
21	next_out	addi r2, r2, #1	000001	5'd2	5'd2	16'd1
22		j outer_loop	001000	5'd0	5'd0	16'd1
23		noop	000000	5'd0	5'd0	16'd0
24	end	j end	001000	5'd0	5'd0	16'd24

2.2.1 Data Processing Type

Instr#	Instr	OP Code [31:26]	Immediate/Offset [25:24]	rs [23:20]	rt [19:16]	rd [15:12]	Offset [11:0]
3	sub r5, r5, #1	000001	10	0101	0101	0001	
6	lsl r8, r8, #2	000011	10	1000	1000	0010	
7	add r9, r4, r8	000000	00	1001	0100	1000	
10	sub r12, r10, r11	000001	00	1100	1010	1011	
14	add r6, r6, #1	000000	10	0110	0110	0001	
15	sub r7, r6, r5	000001	00	0111	0110	0101	

2.2.2 Move Type

Instr#	Instr	OP Code [31:26]	Immediate/Offset [25:24]	Des Reg [23:20]	Sr Reg1 [19:16]	Sr Reg2 [15:12]	Offset [11:0]
4	mov r6, #0	000010	10	0110			
5	mov r8, r6	000010	00	1000	0110		

2.2.3 Load/Store Type

Instr#	Instr	OP Code [31:26]	Des Reg [25:21]	Sr Reg1[20:16]	Sr Reg2[15:0]	Offset[10:0]
	ldr	000100				
	str	000101				

2.2.3 Branch Type

- OP Code Definition

Instr Label	OP Code
add	000000
sub	000001
mov	000010
lsl	000011
ldr	000100
str	000101
b	000110
bge	000111
ble	001000

- Instruction Table

Instr#	Instr	OP Code	Des Reg	Sr Reg1	Sr Reg2	Offset
0	ldr	000100	r4	=array		
1	ldr	000100	r5	=N		
2	ldr	000100	r5	r5		
3	sub	000001	r5	r5	#1	
4	mov	000010	r6			#0
5	mov	000010	r8	r6		
6	lsl	000011	r8	r8		#2
7	add	000110	r9	r4	r8	
8	ldr	000100	r10	r9		
9	ldr	000100	r11	r9		#8
10	sub	000001	r12	r10	r11	
11	ble	001000				
12	str	000101	r11	r9		
13	str	000101	r10	r9		#8
14	add	000000	r6	r6		#1
15	sub	000001	r7	r6	r5	
16	bge	000111				
17	b	000110				
18	b	000110				

```

04 80 10 00 # ldr r4, =array
05 80 20 00 # ldr r5, =N
0A 54 00 00 # ldr r5, [r5]
0B 54 00 01 # sub r5, r5, #1
04 60 00 00 # mov r6, #0
05 68 00 03 # mov r8, r6, LSL #3
06 94 00 00 # add r9, r4, r8
0A A9 00 00 # ldr r10, [r9]
0A B9 00 08 # ldr r11, [r9, #8]
0B C5 A9 00 # sub r12, r10, r11
08 C0 00 20 # ble no_swap
0F B9 00 00 # str r11, [r9]
0F A9 00 08 # str r10, [r9, #8]
04 66 00 01 # add r6, r6, #1

```

2.1.1 Data Processing Instruction

- Instruction Format definition

Condition	00	I	Opcode	S	Rn	Rd	Opprand
31 - 28	27 - 26	25	24 - 21	20	19 - 16	15 - 12	11 - 0

- Condition: Condition Flag
- 00: For Data Processing, Instr[27:26] == 00
- I: Immediate Offset
 - 1 = immediate offset
 - 0 = register offset
- Opcode: ALU_OP code
- S: Set condition flag if S = 1
- Rn: 1st Register addr
- Rd: 2nd Register addr
- Operand: Second operand
 - immediate value or register with shift
- ALU_OP Table

Type	Opcode	Description
ADD	0100	Add
SUB	0010	Subtract
MOV	1101	Move
CMP	1010	Compare
LSL	1101	Logical Left Shift

2.1.2 LDR/STR Instruction

- Instruction Format definition

Condition	01	I	P	U	B	W	L	Rn	Rd	Offset
31 - 28	27 - 26	25	24	23	22	21	20	19 - 16	15 - 12	11 - 0

- Condition: Condition Flags
- Opcode: 01 indicates Load/Store operation
- I: Immediate flag

- 1 = immediate offset
- 0 = register offset
- P: Pre/Post Indexing
 - 1 = Pre
 - 0 = Post
- U: Up/Down
 - 1 = Add offset
 - 0 = Subtract offset
- B: Byte/Word
 - 1 = Byte
 - 0 = Word
- W: Write back to base register
 - 1 = update Rn after access
- L: Load/Store Flag
 - 1 = Load
 - 0 = Store
- Rn: First Register addr
- Rd: Second Register addr
- Offset: Offset
- LDR:
 - 1110, 0101, 1001, 1111, 0011, 0001, 0000, 0100
 - LDR R3, [PC, #4]
 - Condition: 1110
 - 01: 01
 - I: 0
 - P: 1
 - U: 1
 - B: 0
 - W: 0
 - L: 1
 - Rn: 1111
 - Rd: 0011
 - Offset: 000100000100

Condition	01	I	P	U	B	W	L	Rn	Rd	Offset
1110	01	0	1	1	0	0	1	1111	0011	000100000100

- STR:

- 1110, 0101, 0000, 1011, 0011, 0000, 0000, 1100
- STR R3, [FP, #-12]

2.1.3 LDM/STM Instruction

- Instruction Format definition

Condition	100	P	U	S	W	L	Rn	Register List
31 - 28	27 - 25	24	23	22	21	20	19 - 16	15 - 0

- LDM:
 - 1110, 1000, 1011, 1110, 0000, 0000, 0000, 1111
- STM:
 - 1110, 1000, 1000, 1100, 0000, 0000, 0000, 0011

2.2 Generated binary code

- I_MEM.mif

```
e92d4800
e28db004
e24dd038
e59f3104
e24bc038
e1a0e003
e8be000f
e8ac000f
e8be000f
e8ac000f
e89e0003
e88c0003
e3a03000
e50b3008
ea00002e
e51b3008
e2833001
e50b300c
ea000024
e51b300c
e1a03103
e2433004
e083300b
e5132034
e51b3008
e1a03103
e2433004
e083300b
e5133034
e1520003
aa000015
e51b300c
e1a03103
```


e2433004
 e083300b
 e5133034
 e50b3010
 e51b3008
 e1a03103
 e2433004
 e083300b
 e5132034
 e51b300c
 e1a03103
 e2433004
 e083300b
 e5032034
 e51b3008
 e1a03103
 e2433004
 e083300b
 e51b2010
 e5032034
 e51b300c
 e2833001
 e50b300c
 e51b300c
 e3530009
 dafffffd7
 e51b3008
 e2833001
 e50b3008
 e51b3008
 e3530009
 daffffcd
 e3a03000
 e1a00003
 e24bd004
 e8bd4800
 e12fff1e
 0000011c
 00000143
 0000007b
 fffffe39
 00000002
 00000062
 0000007d
 0000000a
 00000041
 fffffffc8
 00000000

•

Hex	Binary	Condition	Opcode	Rn	Rd	Operand
e92d4800	11101001001011010100100000000000	1110	100100	1011	101	100000000000
e28db004	11100010100011011011000000000100	1110	1010	11	110	11000000000100

Hex	Binary	Condition	Opcode	Rn	Rd	Operand
e24dd038	11100010010011011101000000111000	1110	1001	11	111	1000000111000
e59f3104	11100101100111110011000100000100	1110	10110	111	1100	11000100000100
e24bc038	11100010010010111100000000111000	1110	1001	10	1111	111000
e1a0e003	11100001101000001110000000000011	1110	110	1000	11	10000000000011
e8be000f	1110100010111111000000000000001111	1110	100010	1111	1000	1111
e8ac000f	1110100010101100000000000000001111	1110	100010	1011	0	1111
e8be000f	1110100010111111000000000000001111	1110	100010	1111	1000	1111
e8ac000f	1110100010101100000000000000001111	1110	100010	1011	0	1111
e89e0003	1110100010011110000000000000000011	1110	100010	111	1000	11
e88c0003	1110100010001100000000000000000011	1110	100010	11	0	11
e3a03000	11100011101000000011000000000000	1110	1110	1000	0	11000000000000
e50b3008	11100101000010110011000000001000	1110	10100	10	1100	11000000001000
ea00002e	1110101000000000000000000000101110	1110	101000	0	0	101110
e51b3008	111001010000110110011000000001000	1110	10100	110	1100	11000000001000
e2833001	111000101000001100110000000000001	1110	1010	0	1100	11000000000001
e50b300c	11100101000010110011000000001100	1110	10100	10	1100	11000000001100
ea000024	1110101000000000000000000000100100	1110	101000	0	0	100100
e51b300c	111001010000110110011000000001100	1110	10100	110	1100	11000000001100
e1a03103	11100001101000000011000100000011	1110	110	1000	0	11000100000011
e2433004	11100010010000110011000000000100	1110	1001	0	1100	11000000000100
e083300b	11100000100000110011000000001011	1110	10	0	1100	11000000001011
e5132034	1110010100001001100100000000110100	1110	10100	100	1100	10000000110100
e51b3008	111001010000110110011000000001000	1110	10100	110	1100	11000000001000
e1a03103	11100001101000000011000100000011	1110	110	1000	0	11000100000011
e2433004	11100010010000110011000000000100	1110	1001	0	1100	11000000000100
e083300b	11100000100000110011000000001011	1110	10	0	1100	11000000001011
e5133034	1110010100001001100110000000110100	1110	10100	100	1100	110000000110100
e1520003	1110000101010010000000000000000011	1110	101	100	1000	11
aa000015	1010101000000000000000000000010101	1010	101000	0	0	10101
e51b300c	111001010000110110011000000001100	1110	10100	110	1100	11000000001100
e1a03103	11100001101000000011000100000011	1110	110	1000	0	11000100000011
e2433004	11100010010000110011000000000100	1110	1001	0	1100	11000000000100
e083300b	11100000100000110011000000001011	1110	10	0	1100	11000000001011
e5133034	1110010100001001100110000000110100	1110	10100	100	1100	110000000110100
e50b3010	111001010000101100110000000010000	1110	10100	10	1100	11000000010000
e51b3008	111001010000110110011000000001000	1110	10100	110	1100	11000000001000
e1a03103	11100001101000000011000100000011	1110	110	1000	0	11000100000011
e2433004	11100010010000110011000000000100	1110	1001	0	1100	11000000000100
e083300b	11100000100000110011000000001011	1110	10	0	1100	11000000001011
e5132034	1110010100001001100100000000110100	1110	10100	100	1100	10000000110100

Hex	Binary	Condition	Opcode	Rn	Rd	Operand
e51b300c	11100101000110110011000000001100	1110	10100	110	1100	11000000001100
e1a03103	11100001101000000011000100000011	1110	110	1000	0	11000100000011
e2433004	11100010010000110011000000000100	1110	1001	0	1100	11000000000100
e083300b	11100000100000110011000000001011	1110	10	0	1100	11000000001011
e5032034	11100101000000110010000000110100	1110	10100	0	1100	10000000110100
e51b3008	11100101000110110011000000001000	1110	10100	110	1100	11000000001000
e1a03103	11100001101000000011000100000011	1110	110	1000	0	11000100000011
e2433004	11100010010000110011000000000100	1110	1001	0	1100	11000000000100
e083300b	11100000100000110011000000001011	1110	10	0	1100	11000000001011
e51b2010	11100101000110110010000000010000	1110	10100	110	1100	10000000010000
e5032034	11100101000000110010000000110100	1110	10100	0	1100	10000000110100
e51b300c	11100101000110110011000000001100	1110	10100	110	1100	11000000001100
e2833001	11100010100000110011000000000001	1110	1010	0	1100	11000000000001
e50b300c	11100101000010110011000000001100	1110	10100	10	1100	11000000001100
e51b300c	11100101000110110011000000001100	1110	10100	110	1100	11000000001100
e3530009	11100011010100110000000000001001	1110	1101	100	1100	1001
daffffd7	1101101011111111111111111010111	1101	101011	1111	1111	11111111010111
e51b3008	11100101000110110011000000001000	1110	10100	110	1100	11000000001000
e2833001	11100010100000110011000000000001	1110	1010	0	1100	11000000000001
e50b3008	11100101000010110011000000001000	1110	10100	10	1100	11000000001000
e51b3008	11100101000110110011000000001000	1110	10100	110	1100	11000000001000
e3530009	11100011010100110000000000001001	1110	1101	100	1100	1001
daffffcd	11011010111111111111111111001101	1101	101011	1111	1111	11111111001101
e3a03000	11100011101000000011000000000000	1110	1110	1000	0	11000000000000
e1a00003	11100001101000000000000000000011	1110	110	1000	0	11
e24bd004	11100010010010111101000000000100	1110	1001	10	1111	1000000000100
e8bd4800	11101000101111010100100000000000	1110	100010	1111	101	100000000000
e12fff1e	11100001001011111111111100011110	1110	100	1011	1111	11111100011110
0000011c	000000000000000000000000100011100	0	0	0	0	100011100
00000143	000000000000000000000000101000011	0	0	0	0	101000011
0000007b	0000000000000000000000001111011	0	0	0	0	1111011
fffffe39	111111111111111111111111000111001	1111	111111	1111	1111	11111000111001
00000002	00000000000000000000000000000010	0	0	0	0	10
00000062	00000000000000000000000000001100010	0	0	0	0	1100010
0000007d	00000000000000000000000000001111101	0	0	0	0	1111101
0000000a	00000000000000000000000000001010	0	0	0	0	1010
00000041	00000000000000000000000000001000001	0	0	0	0	1000001
fffffc8	11111111111111111111111111001000	1111	111111	1111	1111	11111111001000
00000000	00000000000000000000000000000000	0	0	0	0	0