

Instructions_02

CS2400

Spring 2020

Procedure

- ***Procedure***: A stored subroutine that performs a specific task based on the parameters with which it is provided.
- **Instruction**
- Branch with Link -BL
- Using:
- Link Register
- Program Counter

Register	value
Core	
..... R0	0x00000001
..... R1	0x00000000
..... R2	0x00000003
..... R3	0x00000000
..... R4	0x00000000
..... R5	0x00000000
..... R6	0x00000000
..... R7	0x00000000
..... R8	0x00000000
..... R9	0x00000000
..... R10	0x00000000
..... R11	0x00000000
..... R12	0x00000000
..... R13 (SP)	0x20001000
..... R14 (LR)	0x00000015
..... R15 (PC)	0x00000018
+..... xPSR	0x01000000
+..... Banked	
+..... System	
-..... Internal	
..... Mode	Thread
..... Privilege	Privileged

```

---
19:
→ 0x00000018 EB000302  ADD      r3,r0,r2
20: stop      B stop
<

ifels.s  ifel2.s  loop1.s  fact1.s  Proc1.s
9          EXPORT Reset_Handler
10 Reset_Handler
11 ;~~~~~Usercode here ~~~~~
12
13          MOV  R0,#1
14          MOV  R2,#3
15          BL  add2
16          ADD  R3,R3,#1
17 add2
18          ADD  R3,R0,R2
19
20 stop      B stop
21          END
<

```



Register	Value
Core	
..... R0	0x00000001
..... R1	0x00000000
..... R2	0x00000003
..... R3	0x00000004
..... R4	0x00000000
..... R5	0x00000000
..... R6	0x00000000
..... R7	0x00000000
..... R8	0x00000000
..... R9	0x00000000
..... R10	0x00000000
..... R11	0x00000000
..... R12	0x00000000
..... R13 (SP)	0x20001000
..... R14 (LR)	0x00000015
..... R15 (PC)	0x00000014
+ xPSR	0x01000000
Banked	
System	
Internal	
..... Mode	Thread
..... Privilege	Privileged

Project  Registers

```

16:          ADD R3,R3,#1
17: add2
⇒ 0x00000014 F1030301  ADD      r3,r3,#0x01
18:          ADD R3,R0,R2
<

```

 ifels.s  ifel2.s  loop1.s  fact1.s  Proc1.s

```

10  Reset_Handler
11  ;~~~~~Usercode here ~~~~~
12
13      MOV R0,#1
14      MOV R2,#3
15      BL add2
16      ADD R3,R3,#1
17  add2
18      ADD R3,R0,R2
19      MOV PC ,R14
20
21  stop    B stop
22      END

```

Example Program for procedure call

```

    MOV R0, #1
    MOV R2, #3
    BL proc1
    BL proc2
    MOV R4, R3
    B stop

proc1
    ADD R3, R0, R2
    MOV PC, R14

proc2
    SUB R3, R0, #1
    MOV PC, R14

stop
    B stop

END
```

Q1 -Class work

- //Objective practice on Procedure calling

```
switch (a) {
```

```
    case 1: Procedure call for a=a+1;
```

```
        break;
```

```
    case 2: Procedure call for a=a+2;
```

```
        break;
```

```
    default : Procedure call for a=a+5;
```

```
}
```

Load and Store

- For loads and stores, there are basically two main types of addressing modes available with variations
 - Pre-indexed addressing
 - Post-indexed addressing

Load Addressing Modes

Addressing Mode	Mnemonic	Effective address	Final value in R1
Pre-indexed, Base Unchanged	LDR R0,[R1,#d]	R1+d	R1
Pre-indexed, Base Updated	LDR R0,[R1,#d] !	R1+d	R1+d
Post-indexed, Base Updated	LDR R0,[R1],#d	R1	R1+d

Example address modes

```
''  
    LDR R1, =NUM1  
    LDR R0, [R1, #4]      ; Pre indexed base unchanged  
    ;LDR R0, [R1, #4]!    ; Pre indexed base updated  
    ;LDR R0, [R1], #4     ; Post indexed base updated
```

```
stop B stop
```

```
NUM1 DCD 3, 7, 2, 2, 1  
END
```

```
-----  
..... R0 0x00000007  
..... R1 0x00000010
```

Example address modes Cont..

```
LDR R1, =NUM1
;LDR R0, [R1, #4]      ; Pre indexed base unchanged
LDR R0, [R1, #4]!      ; Pre indexed base updated
;LDR R0, [R1], #4      ; Post indexed base updated
```

stop B stop

```
NUM1 DCD 3, 7, 2, 2, 1
END
```

... R0	0x00000007
... R1	0x00000014

... R2 0x00000000

Example address modes Cont..

```
LDR R1, =NUM1
;LDR R0, [R1, #4]      ; Pre indexed base unchanged
;LDR R0, [R1, #4]!     ; Pre indexed base updated
LDR R0, [R1], #4       ; Post indexed base updated
```

stop B stop

```
NUM1 DCD 3, 7, 2, 2, 1
END
```

...	R0	0x00000003
...	R1	0x00000014

--

- - - - -

Example load addressing modes

__main

```
LDR R1, =arr1
LDR R2, [R1, #4]
LDR R3, [R1, #4] !
LDR R4, [R1], #4
```

st B st

```
arr1 DCD 2, 4, 3, 6, 10, 8, 1
```

END

What will be stored in R2,R3 and R4

1) Loads address of arr1 in R1 ,let it be 100

2)R1 = 100 R2 =4 (Pre-Indexed Base unchanged)

3) R1 = 104 R3=4 (Pre-Indexed Base updated)

4) R1 = 108 R4 =4 (Post-indexed ,Base updated)

• Example Program for Array

```

        LDR R1, N           ; load size of array -
                             ; a counter for how many elem
        LDR R2, POINTER      ; load base pointer of array
        MOV R0, #0          ; initialize accumulator
LOOP
        LDR R3, [R2], #4     ; load value from array,
                             ; increment array pointer to
        ADD R0, R0, R3       ; add value from array to acc
        SUBS R1, R1, #1      ; decrement work counter
        BGT LOOP            ; keep looping until counter
        LDR R4, SUMP         ; get memory address to store
        STR R0, [R4]         ; store answer
SUMP    DCD SUM
N       DCD 5
NUM1    DCD 3, -7, 2, -2, 10
POINTER DCD NUM1

        AREA MYRAM, DATA, READWRITE
SUM     DCD 0
        END
```

String

- Write an assembly program to check if the first letter of a given string is 'H'.

```
LDR R0, = string1
LDRB R2, [R0], #1
LDR R3, =ch;
LDRB R3, [R3]
TEQ R3, R2
```

```
st          B st
string1     DCB "Hello world!", 0
ch          DCB "H"
END
```

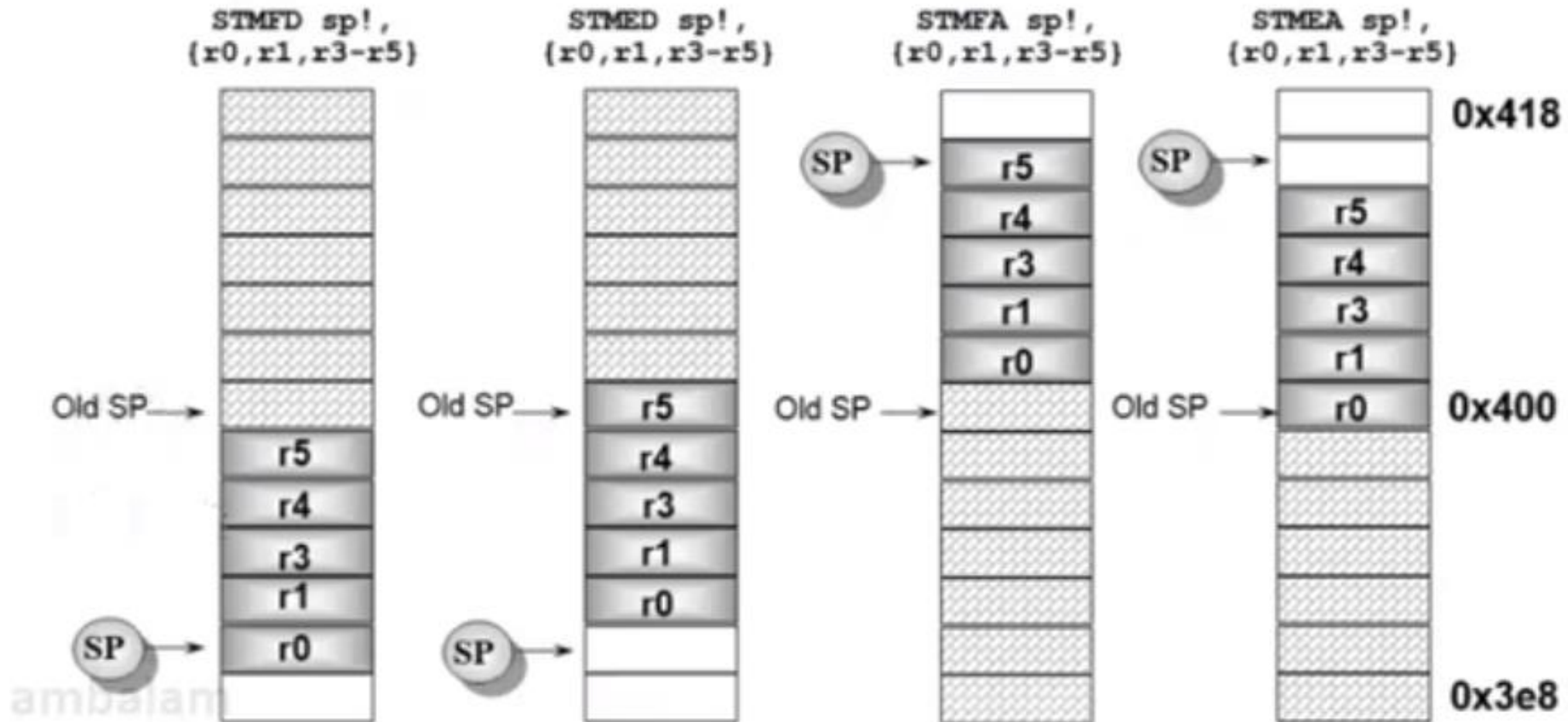
Multiple Register Transfer

- Transfer Block of data
- LDM
- STM
- Syntax **LDM/STM {cond} <addressingmode> Rn{!}, <registers>**



IA – Increment After
IB – Increment Before
DA – Decrement After
DB – Decrement Before

Stack Store



Instruction

- **STMIA** `sp!, {r0-r5}` ; Push onto a Full Descending Stack
LDMDA `sp!, {r0-r5}` ; Pop from a Full Descending Stack

Push /Pop

push `{r1}`

pop `{r3}`

Example

- Program using addressing modes.

	MOV	R0,#1
	MOV	R2,#2
	MOV	R3,#4
	MOV	R4,#6
	LDR	R5,=0x20000018
	MOV	SP,R5 ;Address initialisation for SP
	STMIA	SP!,{R0-R3}
	BL	add2 ;Branch with Link
	ADD	R3,R3,#1
	B	stop
add2	LDMDB	SP!,{R0-R3}
	MOV	R0,#5
	ADD	R3,R0,R2
	MOV	PC ,R14 ; LR in PC
stop	END	

Classwork 2

//Objective to practice on procedure, push and pop.

int a=3,b=6,c=2,d=5; /* use registers to store global variables*/

main{

 checkEvenfun()

 sunFun()

}

checkEvenfun()

{ check if the register have even numbers if odd replace with 0.

 Push values in stack.

}

sunFun()

{ Pop from stack find the sum }

Solution

```
; program stack sum of even numbers
    LDR R0,=arr
    LDR R5,=0x20000018
    BL   pushlabel
    BL   poplabel
    B    stop

pushlabel
    MOV SP,R5
    MOV R4,#0
    MOV R3,#4

iter
    LDR R1,[R0],#4
    TST R1,#1 ; Equal to giving AND R0,R0,#1 and CMP R0
    STMIAEQ SP!,{R1} ; or use pusheq {R1}
    STMIANE SP!,{R4} ;or use pushne {R4}
    CMP R3,#1
    SUBNE R3,R3,#1
    BNE iter
    MOV PC ,R14

poplabel
    MOV R5,#0

loop
    LDMDB SP!,{R1} ; or use pop {r1} ;Address mode should be DB to read back.
    ADD R5,R5,#1
    ADD R4,R4,R1
    CMP R5,#4
    BNE loop
    MOV PC ,R14

stop
arr dcd 2,3,4,1
END
```

Store/Load Multiple

- STMxx R0,{r1-r3}
- LDMxx R0,{r1-r3}

STMIA - LDMxx
STMDA - LDMxx
STMIB - LDMxx
STMDB - LDMxx

Program	R0	r1	r2	r3	R0+0x04	R0+0x08	R0+0x0C
Initial Values	0x100	0x11	0x22	0x33	0x0	0x0	0x0
STMIA R0!,{r1-r3}							
MOV r1,#0							
MOV r2,#0							
MOV r3,#0							
LDMxx R0!,{r1-r3}							

Store/Load Multiple

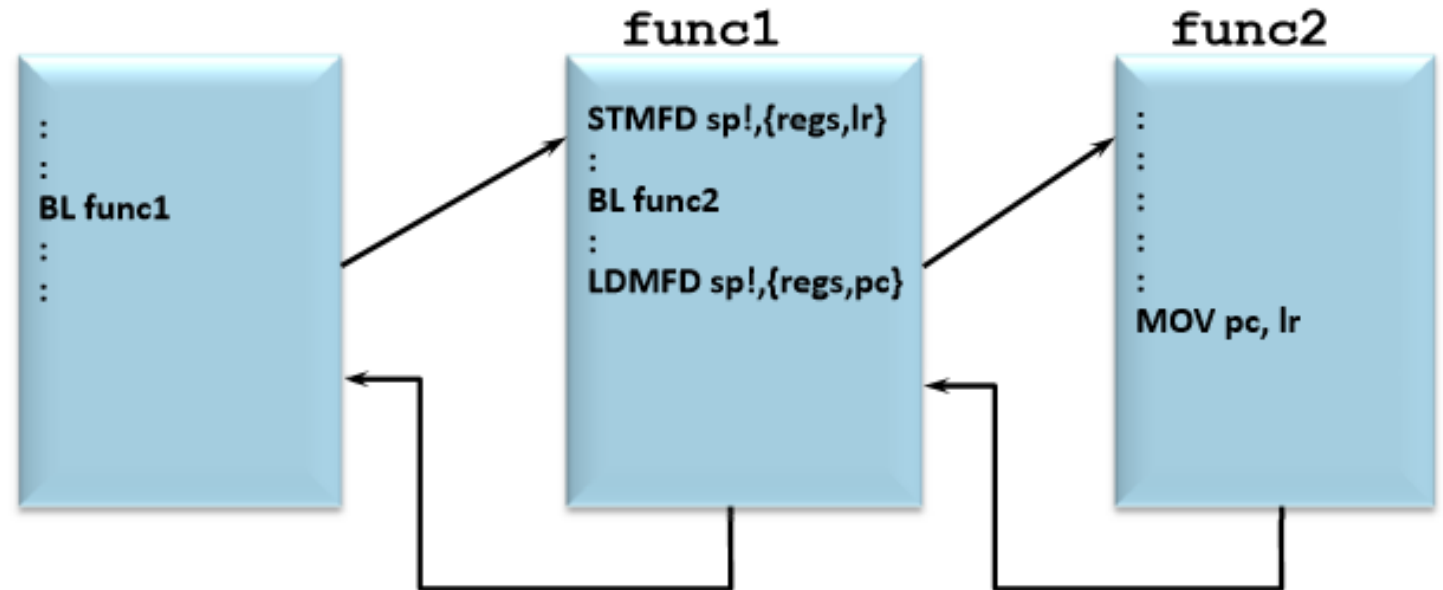
- STMxx R0,{r1-r3}
- LDMxx R0,{r1-r3}

STMIA - LDMDB
STMDA - LDMIB
STMIB - LMDA
STMDB - LDMIA

Program	R0	r1	r2	r3	R0+0x04	R0+0x08	R0+0x0C
Initial Values	0x100	0x11	0x22	0x33	0x0	0x0	0x0
STMIB R0!,{r1-r3}	0x10C	0x11	0x22	0x33	0x11	0x22	0x33
MOV r1,#0	0x10C	0x0	0x22	0x33	0x11	0x22	0x33
MOV r2,#0	0x10C	0x0	0x0	0x33	0x11	0x22	0x33
MOV r3,#0	0x10C	0x0	0x0	0x0	0x11	0x22	0x33
LMDA R0!,{r1-r3}	0x100	0x11	0x22	0x33	0x11	0x22	0x33

Implementing Nested Function

```
main(){
    int a, b;
    a=10; b=5;
    Fun1(); //Function call
    return(0);
}
Fun1(){
    int a;
    a=a+3;
    Fun2(); // Function call
    return();
}
Fun2(){
    int b;
    b=b+1;
    return();
}
```



Nested Function Sample Program

```
; program to simulate nested function
    LDR R0,=a
    LDR R1,=b
    LDR R5,=0x20000018
    BL Fun1
    B stop

Fun1
    MOV R2,#2
    STMFD R5!,{lr}
    BL Fun2
    LDMFD R5!,{pc}

Fun2
    MOV R3,#3
    MOV PC,LR

stop
a DCD 3
b DCD 4
END
```


Try to do Factorial recursively

- **int fact(int n) {
 if (n == 0) {
 return 1;
 } else {
 return n * fact(n - 1);
 }
}**

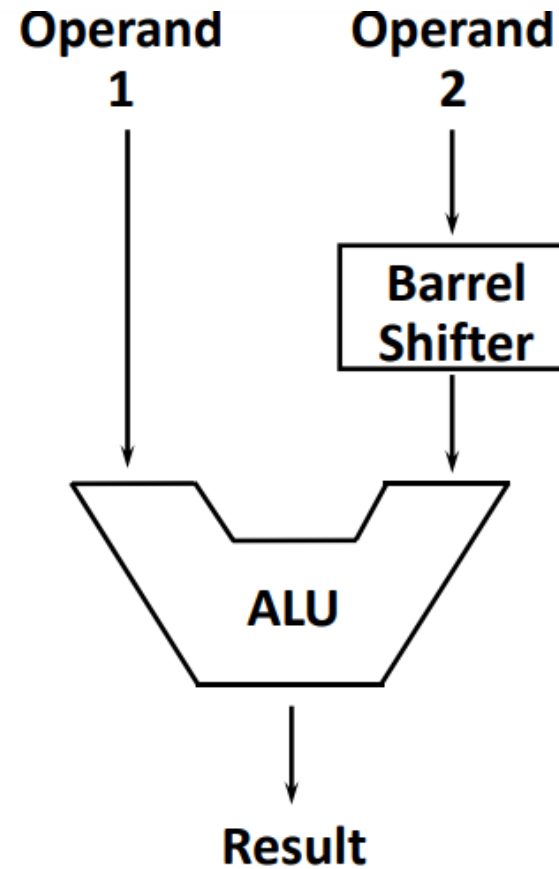
Factorial Solution

```
;; Factorial recursion program
main
    MOV R0, #5
    LDR R4, =0x20000018
    MOV R2, R0
    BL fact
    B finish

fact
    CMP R0, #1
    BEQ exitfun
    STMIA R4!, {lr}
    SUB R0, R0, #1
    MUL R2, R2, R0
    BL fact
exitfun LDMDB R4!, {pc}

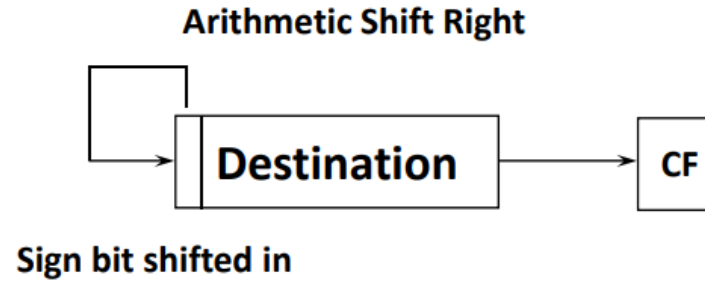
finish
stop    B stop
        END
```

Barrel Shift Process Commands

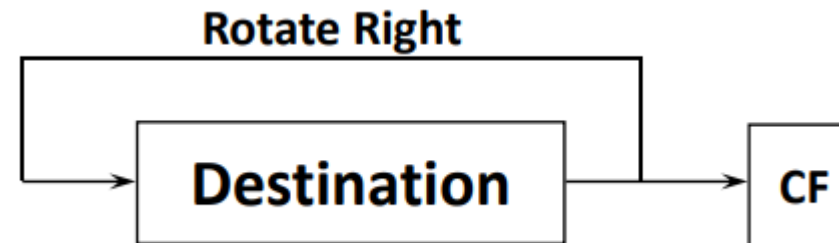


Barrel Shift Process Commands

- **ASR** - Arithmetic Shift Right
- (Sign bit Preserved)



- **ROR**- Rotate Right



Practice before next class

- **MOV R0,#20**
- **MOV R1,#-20**
- **MOV R2, R0, LSL #1**
- **ADD R2, R0, R0, LSR #1**
- **MOV R1, R1, ASR #1**
- **ROR R9, R2, R1, ASR #1**
- **MOV R5, R1, ROR #2**

Instruction Representation

Instruction formats

Instruction	Format
ADD (add)	R
SUB (subtract)	R
ADDI (add immediate)	I
SUBI (sub immediate)	I
LDUR (load register)	D
STUR (store register)	D

opcode	Rm	shamt			(R-type)
	immediate		Rn	Rd/Rt	(I-type)
	address	op2			(D-type)
1112	reg	0	reg	reg	
1624	reg	0	reg	reg	
580	constant		reg	reg	
836	constant		reg	reg	
1986	address	0	reg	reg	
1984	address	0	reg	reg	

Conditional Field

- 0000 = EQ -Z set (equal)
- 0001 = NE -Z clear (not equal)
- 0010 = HS / CS -C set (unsigned higher or same)
- 0011 = LO / CC -C clear (unsigned lower)
- 0100 = MI -N set (negative)
- 0101 = PL -N clear (positive or zero)
- 0110 = VS -V set (overflow)
- 0111 = VC -V clear (no overflow)
- 1000 = HI -C set and Z clear (unsigned higher)
- 1001 = LS -C clear or Z (set unsigned lower or same)
- 1010 = GE -N set and V set, or N clear and V clear (>or =)
- 1011 = LT -N set and V clear, or N clear and V set (>)
- 1100 = GT -Z clear, and either N set and V set, or N clear and V set (>)
- 1101 = LE -Z set, or N set and V clear, or N clear and V set (<, or =)
- 1110 = AL -always
- 1111 = NV -reserved.

Instruction Formats

CORE INSTRUCTION FORMATS

R	opcode	Rm	shamt	Rn	Rd
	31	21 20	16 15	10 9	5 4 0
I	opcode	ALU_immediate		Rn	Rd
	31	22 21		10 9	5 4 0
D	opcode	DT_address	op	Rn	Rt
	31	21 20	12 11 10 9	5 4	0
B	opcode	BR_address			
	31	26 25			0
CB	Opcode	COND_BR_address			Rt
	31	24 23		5 4	0

Example

- ADD R1,R2,R3

- ADD - 0100

- R1 - 0001

- R2 - 0010

- R3 - 0011

- *R-type* Instructions

- Rd –destination

- Rn – R2

- Rm – R3

Opcode	Rm	shamt	Rn	Rd
11 bits	5 bits	6 bits	5 bits	5 bits

Instruction Formats

- *D-type* Instructions

Opcode	address	op2	Rn	Rt
11 bits	9 bits	2 bits	5 bits	5 bits

Instruction Formats

- *I-type* format

Opcode	immediate	Rn	Rd
10 bits	12 bits	5 bits	5 bits

Sample Instructions

	opcode	Rm	shamt	Rn	Rd
ADD X1, X2, X3	1112	3	0	2	1
SUB X1, X2, X3	1624	3	0	2	1

	opcode	immediate	Rn	Rd
ADDI X1, X2, #100	580	100	2	1
SUBI X1, X2, #100	836	100	2	1

	opcode	address	op2	Rn	Rt
LDUR X1, [X2, #100]	1986	100	0	2	1
STUR X1, [X2, #100]	1984	100	0	2	1