ECE243
Kooresh Likes to eat Dirt

1 ARM Assembly

1.1 Definitions

- Word - 32 Bits
- Half-Word - 16 Bits
- Byte - 8 Bits
- Nibble - 4 Bits
- Little Endian - Least significant bit is address of the word. Arm uses this.
- Big Endian - Most significant bit is address of the word
- Machine code is 32 bits wide
- Clobbered Registers - Convention of using registers in arm. Subroutines return in *R0-R1*. The values passed in subroutine should be in *R0-R3*. The subroutine should preserve the state of *R4-R12* upon return.

1.2 Arm Specifics

- Only the following form can be loaded without storing in memory (ror = rotation right).
  - 0-255
  - 256, 260, 264,..., 1020 (Jumps of +4, ROR 30).
  - 1024, 1040, 1056,..., 4080 (Jumps of 16, ROR 28)
  - 4096, 4160, 4224,...,16320 (Jumps of 64, ROR 26)
- Add condition at the end of the command, or *s* for flags (ADDS).

1.3 Registers Used

Registers	Purpose
r0 - r13	General Purpose
r13	Stack (SP)
r14	Link Register (LR)
r15	Program Count (PC)

1.4 CPSR Flags

Condition	Purpose
N	Negative Result
Z	Zero Result
C	Carry flag
V	Overflow flag

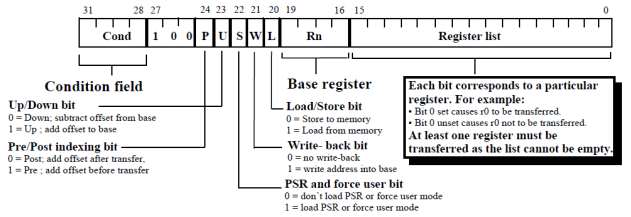
1.5 Condition Codes

Suffix	Condition	Suffix	Condition
EQ	Equal	HI	unsigned GT
NE	!Equal	LS	unsigned LE
HS / CS	unsigned GE	GE	>=
LO / CC	unsigned LT	LT	<
MI	Negative	GT	>
PL	>=0	LE	<=
VS	Overflow	AL	Default
VC	!Overflow	NV	Reserved

1.6 Commands

Op	E.g.	Desc.
	Arithmetic	
ADD/SUB	ADD r0, r1, r2 ADD r0, r1	r0 = r1 ± r2 r0 = r0 ± r1
ADC	ADC r0, r1, r2	r0 = r1 + r2 + c_flag
SBC	SBC r0, r1, r2	r0 = r1 - r2 + !c_flag
MUL	MUL r0, r1, r2	r0 = r1 * r2
MLA	MLA r0, r1, r2, r3	r0 = (r1*r2) + r3
	Comparison	
CMP	CMP r0, r1	Updates condition flags by doing r0-r1
CMN	CMN r0, r1	Updates condition flags by doing r0+r1
TST	TST r0, r1	Updates flags by doing r0 & r1
TEQ	TEQ r0, r1	Update flags by r0 EOR r1
	Logical	
AND ORR EOR	AND r0, r1, r2	r0 = r1 & r2
	Data Movement	
MOV	MOV r0, r1	Move r1 into r0
MVN	MVN r0, r1	Move r1 inverted r0.
	Data Loading	
LDR	LDR r0, [r1] LDR r0, =#0x55555	Load the value pointed by r1 in r0.
LDRB	LDRB r0, [r1, #8]	Load a byte
LDRH	LDRH r0, [r1, #16]	Load a halfword
LDRSB	LDRSB r0, [r1, #4]	Load signed byte
LDRSH	eefwe	Load signed halfword
	Branching	
B	B sub_name	Branch into subroutine
BL	BL sub_name	Branch into func, and save link of return in Link Register
	Shifts	
LSR	LSR r0, r1	Pads 0's in MSB 101011 ->010101
LSL	LSL r0, r1	Pads 0's at the end 101011 ->010110
ASR	ASR r0, r1	Moves MSB same into, r2, and then shifts r1 right. 101011 ->110101
ASL	ASL r0, r1	Pads 0's at the end 101011 ->010110
	Stack - Assembler organizes in increasing order	
PUSH	PUSH {r0, r1, r2, LR} PUSH LR	Push onto stack, note the order does not matter
POP	POP {r1, r2, r0, LR}	Pop from stack, order does not matter.
	Indexing	
Post	LDRB r0, [r1, #4]!	Access r0 at r1, then increment r1 by #4
Pre	LDRB r0, ![r1, #4]	Increment r1 by #4, then access r0 at r1,
Offset	LDRB r0, [r1,#4]	Access r0 at r1+#4

1.7 Operation Code



2 Conversions

2.1 Binary Number

1. Unsigned Binary → Decimal: $\sum_{i=0}^{n-1} b_i \times 2^i$
2. Decimal → Binary: Repetitively divide by 2, round down on odd divisors. Then for each division take the remainders and reverse the order of the remainders
3. 2's Complement → Decimal:
   (a) Check 1st bit. If 0, then convert to decimal regularly.
   (b) If first bit = 1, then start at the right and invert everything **AFTER** the first 1. Then convert to decimal, and add a negative sign before.
4. Decimal → 2's complement:
   (a) Convert the magnitude to binary.
   (b) Start at the right and invert everything **AFTER** the first 1.

2.2 Powers of 2, 8 and 16

n	2	16
0	1	1
1	2	16
2	4	256
3	8	4096
4	16	65536
5	32	1048576
10	1024 (1k)	
11	2048 (2k)	
20	1024 (1M)	
21	2048 (2M)	
30	1024 (1G)	
31	2048 (2G)	
32	4096 (4G)	

3 Our Processor

Function	Purpose
mv rX, rY	rX ← rY
mvi rX, #D	rX ← #D
add rX, rY	rX ← rX + rY
sub rX, rY	rX ← rX - rY
ld rX, [rY]	rX ← [rY]
st rX, [rY]	[rY] ← rX
mvnz rX, rY	if G != 0, rX ← rY
mvnc rX, rY	if no carry-out, rX ← rY

- MVI is two words long
- Bish

ECE243
Kooresh Likes to eat Dirt

```
/* Program that counts consecutive 1's */
.text
.global _start

_start:
MOV R4, #TEST_NUM // test address
MOV R5, #0 // set of 1's
MOV R6, #0 // set of 0's
MOV R7, #0 // set of 01's
LDR R8, =0x55555555 // store the 0101
MOV R9, #0xffffffff // inverting bit

LOOP:
MOV R1, R4 // load the word addr
LDR R1, [R1]
CMP R1,#0 // check last result
BEQ      DISPLAY // if last result, end it

MOV R0, #0 // R0 hold the result
BL ONES // start subroutine ONES
CMP R0, R5 // check the result with the present largest
MOVGE R5, R0 // store the largest 1's

MOV R0, #0 // R0 will hold the result
LDR R1, [R1] // into R1
EOR R1, R9 // invert the bits
BL ZEROES // store the zeroes
CMP R0, R6
MOVGE R6, R0 // store the largest 0's

MOV R0, #0 // R0 will hold the result
LDR R1, [R1] // into R1
EOR R1, R8 //invert with 0101s
EOR R1, R9 // invert the bits
BL ZEROES // store the zeroes
CMP R0, R7
MOVGE R7, R0 // store the largest 01's

ADD R4, #4 // increase r4 address
B LOOP // start the loop again

//Input = R1; Result R = 0
ONES:
CMP R1, #0 // no more 1, keep check
MOVEQ pc, lr // return
LSR R2, R1, #1 // shift until longest 1
AND R1, R1, R2
ADD R0, #1 // count the string length so far
B ONES

//Input = R1; Result R = 0
ZEROES:
CMP R1, #0 // shift until no one left
MOVEQ pc, lr // return
LSR R2, R1, #1 // perform SHIFT, followed by AND
AND R1, R1, R2
ADD R0, #1 // count the string length so far
B ZEROES

END:
B END

/* Display R5 on HEX1-0, R6 on HEX3-2
and R7 on HEX5-4 */
DISPLAY:
LDR R8, =0xFF200020 // base address of HEX3-HEX0

MOV R0, R5 // display R5 on HEX1-0
BL DIVIDE    // ones digit will be in R0;
                          // tens digit in R1
MOV R9, R1           // save the tens digit
BL SEG7_CODE

MOV R4, R0 // save bit code
MOV R0, R9 // retrieve the tens digit and bit code

BL SEG7_CODE

LSL R0, #8
ORR R4, R0

MOV R0, R6 // display R6 on HEX3-2
BL DIVIDE // ones digit will be in R0; tens
                  // digit in R1
MOV R9, R1 // save the tens digit
BL SEG7_CODE

LSL R0, #16 // save the ones digit of r6
ORR R4, R0

MOV R0, R9 // retrieve the tens digit, get bit
                          // code
BL SEG7_CODE

LSL R0, #24
ORR R4, R0

STR R4, [R8] // display the numbers from R6 and R5

LDR R8, =0xFF200030 // base address of HEX5-HEX4


MOV R0, R7 // display R5 on HEX1-0
BL DIVIDE // ones digit will be in R0; tens
                          // digit in R1
MOV R9, R1 // save the tens digit
BL SEG7_CODE

MOV R4, R0 // save bit code
MOV R0, R9 // retrieve the tens digit, get bit
                          // code
BL SEG7_CODE

LSL R0, #8
ORR R4, R0

STR R4, [R8] // display the number from R7

B END

/* Subroutine to convert the digits from 0 to 9 to
*    Parameters: R0 = the decimal value of the digit
*    Returns: R0 = bit patterm to be written to the
*/

SEG7_CODE:
MOV R1, #BIT_CODES
ADD R1, R0 // index into the BIT_CODES "array"
LDRB R0, [R1] // load the bit pattern (to be returne
MOV PC, LR

BIT_CODES:
.byte 0b00111111, 0b00000110,
          0b01011011, 0b01001111,
          0b01100110
.byte 0b01101101, 0b01111101,
          0b00000111, 0b01111111,
          0b01100111
.skip    2 // pad with 2 bytes to maintain word align

/* Subroutine to perform the integer division R0 / R
* Returns: quotient in R1, and remainder in R0
*/
DIVIDE:
MOV R2, #0

CONT:
CMP R0, #10
BLT DIV_END
SUB R0, #10
ADD R2, #1
B CONT

DIV_END:
MOV R1, R2 // quotient in R1 (remainder in R0)
MOV PC, LR

TEST_NUM:
.word 0x103fe00f
.word 0xdd5a62f9
.word 0x298bc0ce
.word 0xb865290d
.word 0x3f32945a
.word 0xdad9fe4b
.word 0xffdadd31
.word 0x0ebb0a11
.word 0x04f57b42
.word 0x746d8ec
.word 0x427b99aa
.word 0x7cfae942
.word 0x0
.end
```