ISA Overview (1A)

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Based on

ARM System-on-Chip Architecture, 2nd ed, Steve Furber

Types of Instructions

Data Processing Instructions
Data Transfer Instructions
Control Flow Instructions

Data Processing Instructions

Arithmetic Operations
Bit-wise Logical Operations
Register Movement Operations
Comparison Operations

Arithmetic Operation

```
ADD     r0, r1, r2      ; r0 := r1 + r2
ADC     r0, r1, r2      ; r0 := r1 + r2 + C
SUB     r0, r1, r2      ; r0 := r1 - r2
SBC     r0, r1, r2      ; r0 := r1 – r2 + C – 1
RSB     r0, r1, r2      ; r0 := r2 + r1
RSC     r0, r1, r2      ; r0 := r2 + r1 + C – 1
```

r0 ⬅ r1 ★ r2

Bit-wise Logical Operations

```
AND     r0, r1, r2      ; r0 := r1 + r2
ADC     r0, r1, r2      ; r0 := r1 + r2 + C
SUB     r0, r1, r2      ; r0 := r1 - r2
SBC     r0, r1, r2      ; r0 := r1 – r2 + C – 1
RSB     r0, r1, r2      ; r0 := r2 + r1
RSC     r0, r1, r2      ; r0 := r2 + r1 + C – 1
```

r0 ⬅ r1 ★ r2

Bit-wise Logical Operations

```
AND     r0, r1, r2      ; r0 := r1 + r2
ADC     r0, r1, r2      ; r0 := r1 + r2 + C
SUB     r0, r1, r2      ; r0 := r1 - r2
SBC     r0, r1, r2      ; r0 := r1 – r2 + C – 1
RSB     r0, r1, r2      ; r0 := r2 + r1
RSC     r0, r1, r2      ; r0 := r2 + r1 + C – 1
```

r0 ⬅ r1 ★ r2

Register Movement Operations

```
MOV     r0, r2          ; r0 := r2
MVN     r0, r2          ; r0 := not r2      Move Negated
```

MOV r0 ⬅ r1

Comparison Operations

CMP	r1, r2	; set cc on r1 – r2	Compare
CMN	r1, r2	; set cc on r1 + r2	Compare Negated
TST	r1, r2	; set cc on r1 and r2	bit Test
TEQ	r1, r2	; set cc on r1 xor r2	Test Equal

r0 ★ r1

Data Transfer Instrucitons

Single Register Load and Store Instructions
Multiple Register Load and Store Instructions
Single Register Swap Instructions

Single Register Load and Store Instructions

```
LDR     r0, [r1]        ; r0 := mem32[r1]
STR     r0, [r1]        ; mem32[r1] := r0

LDR     r0, [r1, #4]    ; r0 := mem32[r1 + 4]
STR     r0, [r1, #4]    ; mem32[r1 + 4] := r0

LDR     r0, [r1, #4]!   ; r0 := mem32[r1 + 4]   ; r1 := r1 + 4
STR     r0, [r1, #4]!   ; mem32[r1 + 4] := r0   ; r1 := r1 + 4

LDR     r0, [r1], #4    ; r0 := mem32[r1]       ; r1 := r1 + 4
STR     r0, [r1], #4    ; mem32[r1] := r0       ; r1 := r1 + 4
```

LDR r0 ⬅ [r1...
STR r0 ➡ [r1...

Multiple Register Load and Store Instructions (1)

```
LDMIA   r1, {r0,r2,r5}      ; r0 := mem32[r1]
                            ; r2 := mem32[r1+4]
                            ; r5 := mem32[r1+8]

LDMIA   r0!, {r2-r4}        ; r2 := mem32[r0], r0 := r0+4
                            ; r3 := mem32[r0], r0 := r0+4
                            ; r4 := mem32[r0], r0 := r0+4

STMIA   r1, {r2-r4}         ; mem32[r1] := r2
                            ; mem32[r1+4] := r3
                            ; mem32[r1+8] := r4
```

LDM r0 ⬅ {r1...}
STM r0 ➡ {r1...}

Multiple Register Load and Store Instructions (2)

STMFD r5! {r2-r3} ; r5 := r5-4 ; mem_{32}[r5] := r2
; r5 := r5-4 ; mem_{32}[r5] := r3

LDMIA r0!, {r2-r3} ; r2 := mem_{32}[r0] ; r0 := r0+4
; r2 := mem_{32}[r0] ; r0 := r0+4

STMIA r1, {r2-r3} ; mem_{32}[r1] := r2
; mem_{32}[r1+4] := r3

LDMFD r5!, {r2-r3} ; r2 := mem_{32}[r5] ; r5 := r5+4
; r3 := mem_{32}[r5] ; r5 := r5+4

LDM r0 ⬅ {r1...}
STM r0 ➡ {r1...}

Multiple Register Load and Store Instructions (3)

```
STMFD  r5! {r2-r3}          ; save regs onto stack
LDMIA   r0!,  {r2-r3}
STMIA   r1,   {r2-r3}
LDMFD r5!, {r2-r3}          ; restore regs from stack
```

FD Stack

STORE	PUSH	Growing Downward	Filled TOP
LOAD	POP	Contracting Upward	Filled TOP

IA Block Copy

STORE	PUSH	Growing Upward	Empty TOP
LOAD	POP	Contracting Downward	Empty TOP

LDM r0 ⬅ {r1...}
STM r0 ➡ {r1...}

Single Register Swap Instruction

```
ADR     r0, SEMAPHORE
SWAP    r1, r1, [r0]          ; exchange byte
```

Control Flow Instructions

Branch Instruction
Branch and Link Instruction
Subroutine Return Instruction
Supervisor Call Instruction

Branch Instructions

```
        B    LABEL
        …
LABEL   …


        BL   SUBR          ; branch to SUBR
        …                  ; return to here
SUBR    …                  ; subroutine entry point
        MOV      pc, r14   ; return
```

Branch and link Instruction

```
        BL   SUBR           ; branch to SUBR
        …                   ; return to here
SUBR    …                   ; subroutine entry point
        MOV      pc, r14    ; return


        BL   SUB1
        …
SUB1    STMFD  r13!, {r0-r2, r14}
        BL   SUB2
        …
SUB2
```

Subroutine return address

```
SUB2    …
        MOV     pc, r14         ; copy r14 into r15 to return

SUB1    STMFD   r13!, {r0-r2,r14};   save work regs  and link
        BL      SUB2
        …
        LDMFD   r13!, {r0-r2,pc}
```

Supervisor calls

```
SWI     SWI_WriteC      ; output r0[7:0]

SWI     SWI_Exit        ; return to monitor
```

Supervisor calls

```
SWI     SWI_WriteC      ; output r0[7:0]

SWI     SWI_Exit        ; return to monitor
```

Supervisor Branch conditions

B	Unconditional	Always take this branch
BAL	Always	Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not Equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC	Carry Clear	Arithmetic operation did not give carry-out
BLO	Lower	Unsigned comparison gave lower
BCS	Carry Set	Arithmetic operation gave carry-out
BHS	Higher or Same	Unsigned comparison gave higher or same
BVC	Overflow Clear	Signed integer operation; no overflow occurred
BVS	Overflow Set	Signed integer comparison; overflow occurred
BGT	Greater Than	Signed integer comparison gave greater than
BGE	Greater or Equal	Signed integer comparison gave greater or equal
BLT	Less Than	Signed integer comparison gave less than
BLE	Less or Equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Unsigned comparison gave lower or same

References

[1] ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf
[2] https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf