



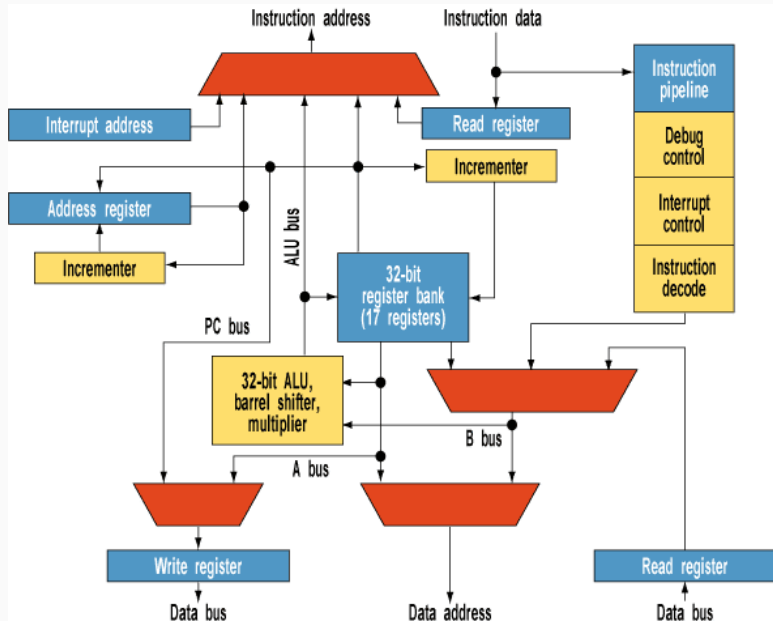
Sequence 5.2 – ARM Assembly

P. de Oliveira Castro S. Tardieu

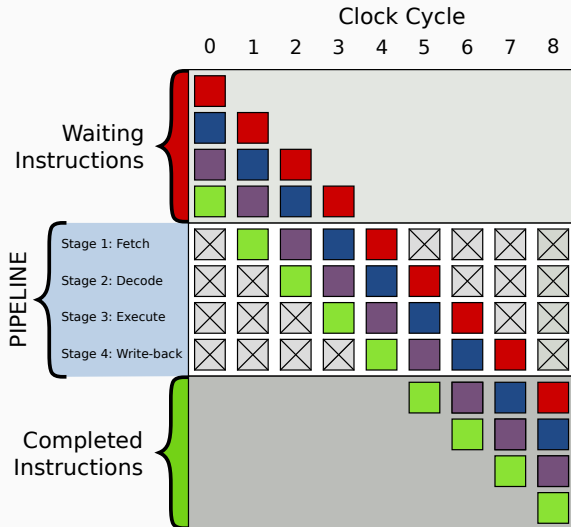
Introduction

- First developed by Acorn computers in 1983
- Simple and versatile architecture
- RISC (Reduced Instruction Set Computer)
 - Fixed size instructions, 16 bits in thumb mode (nice for embedded systems)
 - Modified Harvard pipelined architecture (instruction and data cache separated)

Micro-architecture



Pipeline



- Thumb mode:
 - 8 general purpose registers (32 bits) R0-R7
 - PC: program counter
 - SP: stack pointer
 - LR: link register
 - Usually R7 is used as a frame pointer (FP)

Encoding thumb instructions

- The following 16-bits encode a thumb instruction, 00011000
10001000
- Instead of directly generating bytes, we use *mnemonics*
`adds r0, r1, r2 # r0 <- r1 + r2`
- The *assembler* transforms a list of mnemonics into binary code

Additions

- Adds the values in registers `r1` and `r2` and stores the result in register `r0`
- The `s` updates the *status flag register* which tells if a result is positive, null, or overflows.

```
adds r0, r1, r2
```


Arithmetic operations

- `add{s} r1, r2, r3` $\implies r1 \leftarrow r2 + r3$
- `sub{s} r1, r2, r3` $\implies r1 \leftarrow r2 - r3$
- `mul{s} r1, r2, r3` $\implies r1 \leftarrow r2 \times r3 \ (r3 \neq r1)$

- `and`, `or`, `eor` encode respectively `and`, `or` and `xor`
- `tst r0, r1` performs a logical `and` between `r0` and `r1` without updating the flag register.

Immediate values

- `ADD r0, r0, #1` increments register `r0`
- The range of immediates is limited due to the 16 bit encoding constraint

Comparing registers

- `CMP r0, r1`
- `CMP r0, #5`
- Subtracts `r0` and `r1` (or `#5`) and updates the flag register

- Unconditional branches jump to a different program position

```
infinite_loop:  
    add r1, r1, #1  
    b infinite_loop
```

- Conditional branches depend on the flag register state

`beq label` @ branch if last CMP was equal

`bne label` @ branch if last CMP was different

`blt label` @ branch if last CMP was lower than

`...`

Example: if / then / else in assembly

```
if ( r0 < 0) then /* then */ else /* else */
```

```
    CMP r0, #0
```

```
    BGE else
```

```
    ..then..
```

```
    B out
```

```
else:
```

```
    ..else..
```

```
out:
```

Example: Computing $\text{pow}(m,n)$

- Program computing $\text{pow}(m,n)$
 - Register r1 contains positive integer n
 - Register r2 contains positive integer m .
 - Do not worry about overflow problems here.
- Write an assembly program to compute $\text{pow}(m,n)$.

Example: Computing pow(m,n)

```
    mov r0, #1
    cmp r1, #0
mult:
    beq out
    mul r0,r0,r2
    subs r1,r1,#1
    b mult
out:
```

- Memory is addressed by bytes
 - LDR/STR load or store a word (32 bits)
 - LDRH/STRH load or store a half-word (16 bits)
 - LDRB/STRB load or store a byte (8 bits)
 - LDRSB/STRSB load or store and extend sign

- `ldr r0, [r1]` load in register r0 the value at address r1
- `str r0, [r1, #16]` write at address r1+16 the value in r0

The stack

- Usually in ARM the stack grows downwards
- `sp` always points to the top of the stack
- `pop` and `push` allow to pop and push registers to the stack

Branch and link

- To call a function in assembly we jump to its label
- Unlike a simple branch, we must keep the return address
- The mnemonic `bl label`:
 - Jumps to *label*
 - Stores in the `lr` link register the return address
- To get back to the caller we can use the following mnemonic,

`bx lr`