University of BRISTOL

Overview of Computer Architecture

Further Assembly Programming

Carl Henrik Ek - carlhenrik.ek@bristol.ac.uk

November 15, 2019

http://carlhenrik.com

Today

- Short recap
- More assembler programming

Recap

Assembly Code

Machine Code

- `0xe3a0002a`
- `0xe0211001`
- `0xe0811000`
- `0xeafffffb`

Mnemonics

- `mov r0, #42`
- `eor r1, r1, r1`
- `add r1, r1, r0`
- `b _start`

Assembler

- The assembler is just a "stupid" translator, it converts `mnemonics` to binary

Assembler

- The assembler is just a "stupid" translator, it converts `mnemonics` to binary
- Resolves addresses

Assembler

- The assembler is just a "stupid" translator, it converts `mnemonics` to binary
- Resolves addresses
  - `_start` $\Rightarrow$ `0x8000`

Assembler

- The assembler is just a "stupid" translator, it converts `mnemonics` to binary
- Resolves addresses
  - `_start` $\Rightarrow$ `0x8000`
- Re-writes code to be PC-relative

Assembler

- The assembler is just a "stupid" translator, it converts `mnemonics` to binary
- Resolves addresses
  - `_start` $\Rightarrow$ `0x8000`
- Re-writes code to be PC-relative
  - `ldr r0, _start` $\Rightarrow$ `ldr r0, [pc,#40]`

Assembler

- The assembler is just a "stupid" translator, it converts `mnemonics` to binary
- Resolves addresses
  - `_start` $\Rightarrow$ `0x8000`
- Re-writes code to be PC-relative
  - `ldr r0, _start` $\Rightarrow$ `ldr r0, [pc,#40]`
- Organises code into block

Loops

Code

```
        mov         r7, #42-1
_loop:
        subs         r7, r7, #1
        bne         _loop
```

Functions

Code

```
        ....
        bl          _sub    @ call _sub
        ....                @ will return here

_sub:
        stmdb           sp!, {r2-r3,r7}

        ldmia           sp!, {r2-r3,r7}
        mov          pc, r14
```

Addressing Modes

Register to register `MOV R0, R1`

Addressing Modes

Register to register `MOV R0, R1`

Absolute `LDR R0, 0x12345`

Addressing Modes

Register to register `MOV R0, R1`

Absolute `LDR R0, 0x12345`

Literal `MOV R0, #15`

Addressing Modes

Register to register `MOV R0, R1`

Absolute `LDR R0, 0x12345`

Literal `MOV R0, #15`

Indexed, base `LDR R0, [R1]`

Addressing Modes

Register to register `MOV R0, R1`

Absolute `LDR R0, 0x12345`

Literal `MOV R0, #15`

Indexed, base `LDR R0, [R1]`

Pre-indexed `LDR R0, [R1, #4]`

Addressing Modes

Pre-indexed with increment `LDR R0, [R1, #4]!`

Addressing Modes

Pre-indexed with increment `LDR R0, [R1, #4]!`

Post-indexing with post increment `LDR R0, [R1], #4`

Addressing Modes

Pre-indexed with increment `LDR R0, [R1, #4]!`

Post-indexing with post increment `LDR R0, [R1], #4`

Double Reg indirect `LDR R0, [R1, R2]`

Addressing Modes

Pre-indexed with increment `LDR R0, [R1, #4]!`

Post-indexing with post increment `LDR R0, [R1], #4`

Double Reg indirect `LDR R0, [R1, R2]`

Double Reg indirect with shift `LDR R0, [R1, r2, LSL #2]`

Addressing Modes

Pre-indexed with increment `LDR R0, [R1, #4]!`

Post-indexing with post increment `LDR R0, [R1], #4`

Double Reg indirect `LDR R0, [R1, R2]`

Double Reg indirect with shift `LDR R0, [R1, r2, LSL #2]`

Program counter relative `LDR R0, [PC, #offset]`

Example Code

Example Code

1. Loops
    - find the largest number in a sequence
2. Compare two strings
    - cases
3. Recursion
    - compute factors of numbers
4. Secret

Summary

Summary

- We have seen most of execution and assembly programming now
- We have only seen the `CPU`
- Practice makes perfecta
- Post explore week
  - Lecture by Kerstin
  - One more lab on assembly programming
- Week 10 we will continue with new material

eof