Elementary ARM for Reversing

Claud Xiao

Why learning?

- Most of mobile platforms are based on ARM.
  - Android
  - Symbian
  - Windows Phone 7
  - Apple iOS
- As well as some network devices.
- New trends in malware's evolution

What’s ARM

- A company
- CPU products
- Instruction architectures

Versions

Products	Instruction sets
ARM7	ARM v4
ARM9	ARM v5
ARM11	ARM v6
Cortex-A	ARM v7-A
Cortex-R	ARM v7-R
Cortex-M	ARM v7-M

Architecture

- 32 bits
- RISC
- von Neumann architecture
  - Mixed data and code
  - ARM7
- Harvard architecture
  - Separated data and code
  - ARM9 and later

Registers

- **`R0`** – **`R3`** (a1 – a4)
  - Common registers
  - For argument passing and result return
- **`R4`** – **`R11`** (v1 – v8)
  - Common registers
  - For local variable
- **`R12`** (IP)
  - Intra-Procedure-call scratch register

Registers (cont.)

- R13 (**SP**)
  - Stack Pointer
- R14 (**LR**)
  - Link Register
- R15 (**PC**)
  - Program Counter

Registers (cont.)

- PSRs: Program Status Registers
  - ALU flags (zero flag and carry flag)
  - execution status
  - current executing interrupt number

Data instructions

MOV*cd***S** reg, arg

CMP*cd* reg, arg

ADD*cd***S** reg, reg, arg

Form:

```
<opcode>{<cond>}{S} <Rd>, <Rn>{, <operand2>}
```

Data instructions (cont.)

- `opcode`: **`MOV, ADD, CMP`** ...
- `cond`: **`EQ, NE, GE, LT`** ...
- **`S`**: update PSR flags
- `reg`: **`R0`** – **`R15`**
- `arg`: right-hand argument (next page)

Forget ALL above!

Just lookup references when using them.

Right-hand argument

- `#imm`$_8$: immediate (rotated into 8 bits)
  - **`MOV R0, #12`**
- `reg`: register
  - **`ADDS R0, R0, R1`**
- `reg, shift`: register shifted by distance
  - **`CMP R0, R1, LSL #2`**
  - **`ADDNE R0, R0, R1, ASR R3`**

Branch instructions

- `B`*`cd`* `imm`$_{12}$
  - Branch to imm$_{12}$ words away
- `BL`*`cd`* `imm`$_{12}$
  - Copy PC to LR, then branch
- `BX`*`cd`* `reg`
  - Copy reg to PC

Memory instructions

- `**LDR***cd***B** reg, *mem*`
  - Loads word/bytes from memory
- `**STR***cd***B** reg, *mem*`
  - Stores word/bytes to memory
- **B**: operate by byte instead of by word
- *mem*: memory address (next page)

Memory address

- `[reg, #±imm`$_{12}$`]!`
  - `reg` offset by constant
- `[reg, ±reg]!`
  - `reg` offset by variable bytes
- `[reg`$_a$`, ±reg`$_b$`, `*`shift`*`]!`
  - `reg`$_a$ offset by shifted variable `reg`$_b$

- `!`: update `reg`, then access memory

Memory address(cont.)

- `[reg], #±imm`$_{12}$
  - Access address `reg`, then update `reg` by offset
- `[reg], ±reg`
  - Access address `reg`, then update `reg` by variable
- `[reg], ±reg, shift`
  - Access address `reg`, then update `reg` by shifted variable

Memory instructions(cont.)

- **`LDM`**`*cdum* reg!, *mreg*`
  - Loads memory in `reg` into multiple registers
- **`STM`**`*cdum* reg!, *mreg*`
  - Stores multiple registers into memory in `reg`

- *um*: update mode (next page)
- `!`: change the register
- *mreg*: e.g. **`{R0-R3, R7, R10}`**

Update mode

- **`IA`**: address Increment After each read
- **`DA`**: … Decrement After …
- **`IB`**: … Increment Before …
- **`DB`**: … Decrement Before …

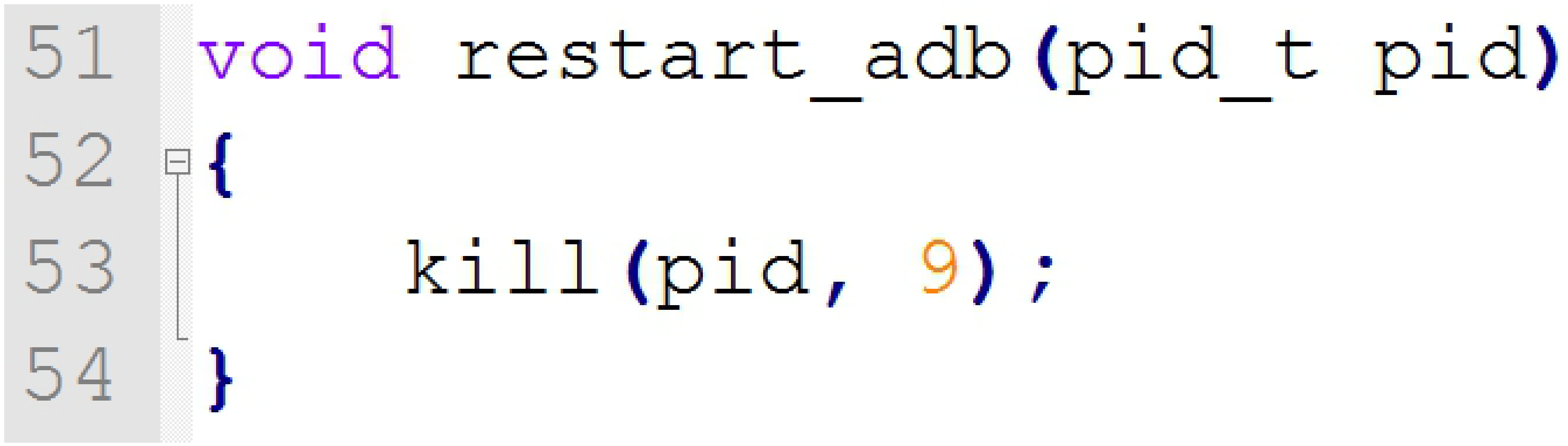
Stack model

- Full Descending stack
- Push: `STMFD SP!, mreg ; ==STMDB`
  - `SUB SP, SP, 4`
  - `MOV Rn, [SP]`
- Pop: `LDMFD SP!, mreg ; ==LDMIA`
  - `MOV Rn, [SP]`
  - `ADD SP, SP, 4`

More ATPCS

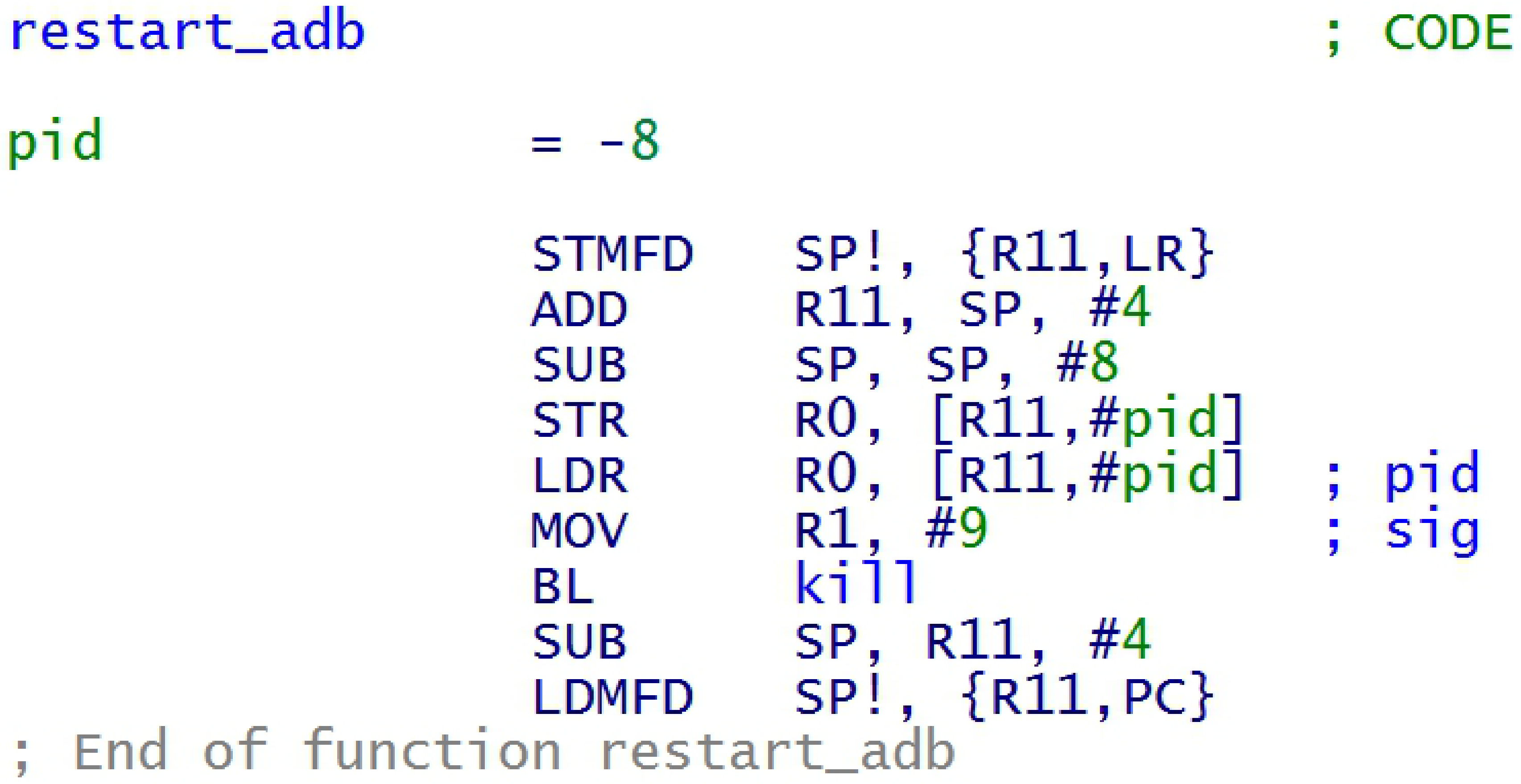
- Parameter passing
  - Use **`R0`** – **`R3`** (a1 – a4) firstly, copy from left to right;
  - Use stack if more than 4, push from right to left.
- Result return
  - Use **`R0`** – **`R3`** (a1 – a4) firstly
  - Use an additional address parameter if 4 registers is not enough

Example



```
void restart_adb(pid_t pid)
{
    kill(pid, 9);
}
```

Example(cont.)



```
restart_adb                                           ; CODE

pid             = -8

                STMFD   SP!, {R11,LR}
                ADD     R11, SP, #4
                SUB     SP, SP, #8
                STR     R0, [R11,#pid]
                LDR     R0, [R11,#pid]       ; pid
                MOV     R1, #9               ; sig
                BL      kill
                SUB     SP, R11, #4
                LDMFD   SP!, {R11,PC}
; End of function restart_adb
```

return 0;

Thank you!

Q & A