Microprocessors & Microcontrollers

: Arm Cortex M0+

(Using RP2040)

ESM_14

Writing Assembly Programs

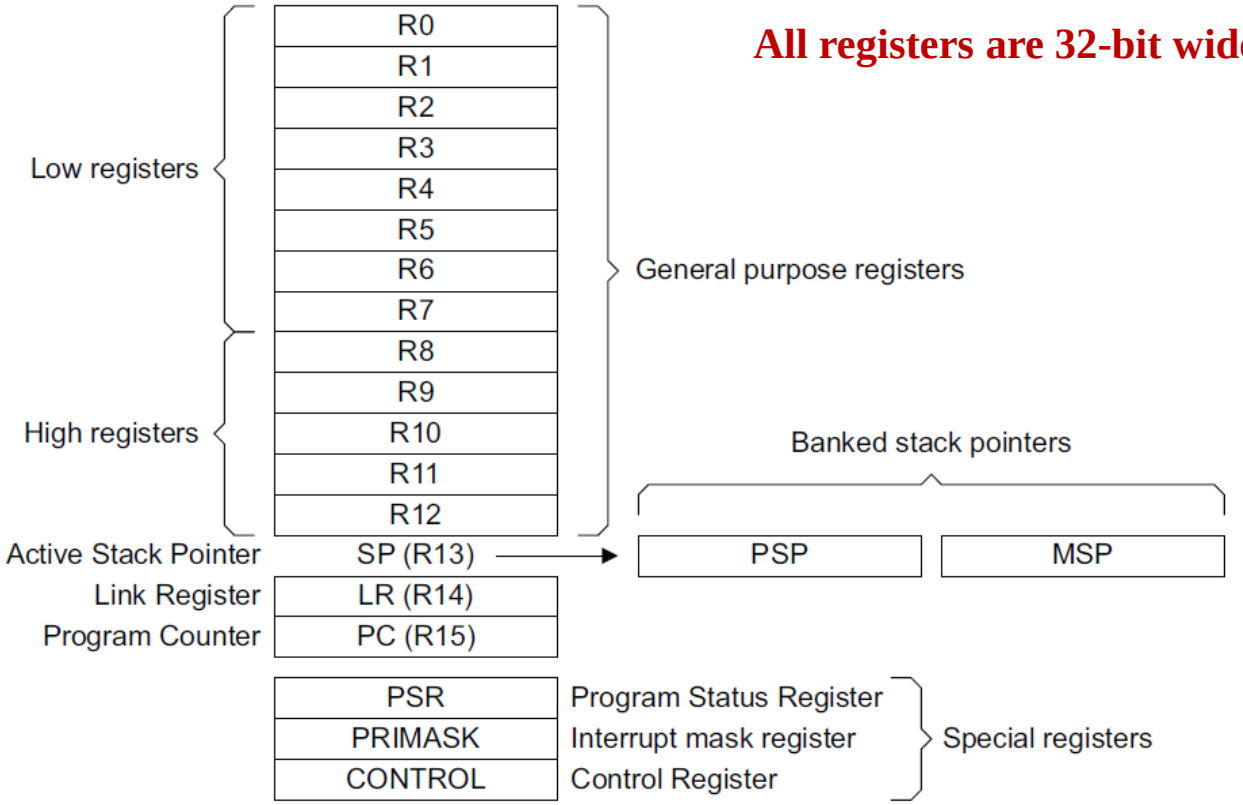
Focus

- Registers in Arm Cortex-M0+
- Assembly Syntax
  - **mov** and **bx** instructions
- Demo of **Asm** program

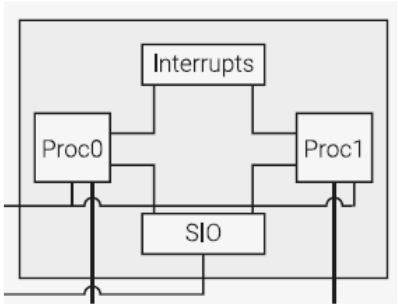
Ref: Ref3_ARM-Cortex-M0+ Technical Reference Manual.pdf

Registers in Arm Cortex M0+

Arm-Cortex-M0+: Register Set



All registers are 32-bit wide



PSP: Process Stack Pointer

MSP: Main Stack Pointer (default on reset)

Ref: Ref4_ARM-Cortex-M0+ Devices Generic User Guide, Section 2.1.3 Core Registers

Assembly Syntax

Calling an asm function from C

prog_lec_2_2_2.ino | diagram.json | main_asm.S | Library Manager

```
13  #include<stdio.h>
14  #include<stdlib.h>
15
16  // Uncomment this if when you copy this code to run it on HW
17  //#define RUN_ON_HW
18
19  #ifdef RUN_ON_HW
20  #define Serial1 Serial
21  #endif
22
23  void myPrint(char* fnName, int result);
24
25  // Define external assembly function
26  extern "C" {
27    unsigned int my_asm_fn(void);
28  }
```

This is part of the program being demonstrated in this lecture:
prog_lec_2_2_2.ino

.S is the extension for the assembly program file.

```
7   @ Ref: ARMv6M Technical reference to learn about the
8   @ Syntax and usage of asm instructions
9   // Label of the asm function
10  .global my_asm_fn   @ To provide the address of this la
11
12  .text
13  @ *******
14  // This function just returns  value of 200 through r0
15  my_asm_fn:
16   mov r0, #200    @ Return a value of 200 in decimal back
17   bx lr           @ Return from the asm function back to
18
19  .end
20
21  @ End of text or code segment
```

Assembly Syntax: main_asm.S

- Comments: Starting a line with **@** symbol or double slash //
- **.text** is the starting of the code segment, where assembly instructions are stored
- Labels end with a **colon :**
- **.end** marks the end of a section
- Immediate constants start with a #
- *mov r0, #200*
- Moves the value 200 in decimal into the register **r0**
- bx lr
- **Bx: Branch with exchange**
- **LR** is the link register (**R14**) which has the code address where the function should return back

```
7   @ Ref: ARMv6M Technical reference to learn about the
8   @ Syntax and usage of asm instructions
9   // Label of the asm function
10  .global my_asm_fn   @ To provide the address of this la
11
12  .text
13  @ *******
14  // This function just returns  value of 200 through r0
15  my_asm_fn:
16   mov r0, #200   @ Return a value of 200 in decimal back
17   bx lr          @ Return from the asm function back to
18
19  .end
20
21  @ End of text or code segment
```

This is part of the program being demonstrated in this lecture: **prog_lec_2_2_2.ino**

.S is the extension for the assembly program file.

Instruction Set

The processor implements the ARMv6-M Thumb instruction set, including a number of 32-bit instructions that use Thumb-2 technology. The ARMv6-M instruction set comprises:

- All of the 16-bit Thumb instructions from ARMv7-M excluding CBZ, CBNZ and IT.
- The 32-bit Thumb instructions BL, DMB, DSB, ISB, MRS and MSR.

Each of the following sections describes a functional group of Instructions of Cortex-M0+ instructions:

- Memory access instructions: eg: ADR, LDM, PUSH, POP, STR, etc
- General data processing instructions: eg: ADCS, ADD, MULS, CMP, LSRS, LSLS, SUBS, SUCS etc
- Branch and control instructions: eg: BL, BLX, BX,etc
- Miscellaneous instructions: eg: SB, MRS, MSR, NOP, SEV, SVC, etc

List of Instruction set supported by Arm Cortex M0+

1. ADC - Add with Carry
2. ADD - Add
3. AND - Bitwise AND
4. B - Branch
5. BEQ - Branch if Equal
6. BNE - Branch if Not Equal
7. BX - Branch and Exchange
8. BL - Branch with Link
9. CMP - Compare
10. CMN - Compare Negative
11. LDR - Load Register
12. LDRB - Load Register Byte
13. LDRH - Load Register Halfword
14. LDRSH - Load Register Signed Halfword
15. LDRSB - Load Register Signed Byte
16. STR - Store Register
17. STRB - Store Register Byte
18. STRH - Store Register Halfword
19. MOVS - Move and Set Flags
20. MOV - Move
21. MUL – Multiply
22. MVN - Move Not
23. NOP - No Operation
24. ORR - Bitwise OR
25. POP - Pop Stack
26. PUSH - Push Stack
27. RSB - Reverse Subtract
28. SBC - Subtract with Carry
29. SUB - Subtract

List of Instruction set supported by Arm Cortex M0+

30. SVC - Supervisor Call
31. TST - Test
32. TEQ - Test Equivalent
33. ASR - Arithmetic Shift Right
34. BIC - Bit Clear
35. CLZ - Count Leading Zeros
36. EOR - Exclusive OR
37. IT - If Then
38. LSL - Logical Shift Left
39. LSR - Logical Shift Right
40. ROR - Rotate Right
41. RSB - Reverse Subtract
42. RSC - Reverse Subtract with Carry
43. REV - Byte-Reverse Word
44. REV16 - Byte-Reverse Word Halfword
45. REVSH - Byte-Reverse Signed Halfword
46. SXTB - Sign Extend Byte
47. SXTH - Sign Extend Halfword
48. UXTB - Unsigned Extend Byte
49. UXTH - Unsigned Extend Halfword
50. WFE - Wait For Event
51. WFI - Wait For Interrupt
52. SEV - Send Event
53. CPSID - Disable Interrupts
54. CPSIE - Enable Interrupts
55. BKPT - Breakpoint
56. BKPT #imm - Breakpoint with Immediate Value

Technical Reference Manual of Cortex-M0+ (Ref 3)

3.3 Instruction set summary

The processor implements the ARMv6-M Thumb instruction set, including a number of 32-bit instructions that use Thumb-2 technology. The ARMv6-M instruction set comprises:

- All of the 16-bit Thumb instructions from ARMv7-M excluding `CBZ`, `CBNZ` and `IT`.
- The 32-bit Thumb instructions `BL`, `DMB`, `DSB`, `ISB`, `MRS` and `MSR`.

Table 3-1 shows the Cortex-M0+ instructions and their cycle counts. The cycle counts are based on a system with zero wait-states.

Table 3-1 Cortex-M0+ instruction summary

Operation	Description	Assembler	Cycles
Move	8-bit immediate	`MOVS Rd, #<imm>`	1
	Lo to Lo	`MOVS Rd, Rm`	1
	Any to Any	`MOV Rd, Rm`	1
	Any to PC	`MOV PC, Rm`	2
Add	3-bit immediate	`ADDS Rd, Rn, #<imm>`	1
	All registers Lo	`ADDS Rd, Rn, Rm`	1
	Any to Any	`ADD Rd, Rd, Rm`	1
	Any to PC	`ADD PC, PC, Rm`	2
	8-bit immediate	`ADDS Rd, Rd, #<imm>`	1
	With carry	`ADCS Rd, Rd, Rm`	1
	Immediate to SP	`ADD SP, SP, #<imm>`	1

Arithmetic Instructions

Syntax: <instruction>{<cond>}{S} Rd, Rn, N

ADC	add two 32-bit values and carry	$Rd = Rn + N +$ carry
ADD	add two 32-bit values	$Rd = Rn + N$
RSB	reverse subtract of two 32-bit values	$Rd = N - Rn$
RSC	reverse subtract with carry of two 32-bit values	$Rd = N - Rn -$!(carry flag)
SBC	subtract with carry of two 32-bit values	$Rd = Rn - N -$!(carry flag)
SUB	subtract two 32-bit values	$Rd = Rn - N$

Problems

```
PRE     r0 = 0x00000000
        r1 = 0x00000002
        r2 = 0x00000001

        SUB r0, r1, r2

POST    r0 = 0x00000001
```

```
PRE     r0 = 0x00000000
        r1 = 0x00000077

        RSB r0, r1, #0      ; Rd = 0x0 - r1

POST    r0 = -r1 = 0xffffff89
```

Problems

EXAMPLE 3.4 This simple subtract instruction subtracts a value stored in register *r2* from a value stored in register *r1*. The result is stored in register *r0*.

```
PRE     r0 - 0x00000000
        r1 - 0x00000002
        r2 - 0x00000001

        SUB r0, r1, r2

POST    r0 - 0x00000001
```

EXAMPLE 3.6 The SUBS instruction is useful for decrementing loop counters. In this example we subtract the immediate value one from the value one stored in register *r1*. The result value zero is written to register *r1*. The *cpsr* is updated with the *ZC* flags being set.

```
PRE     cpsr - nzcvqiFt_USER
        r1 - 0x00000001

        SUBS r1, r1, #1


POST    cpsr - nZCvqiFt_USER
        r1 - 0x00000000
```

Problems

EXAMPLE 3.34 This example shows an ADD instruction with the EQ condition appended. This instruction will only be executed when the zero flag in the *cpsr* is set to 1.

```
; r0 = r1 + r2 if zero flag is set
ADDEQ r0, r1, r2
```

Only comparison instructions and data processing instructions with the S suffix appended to the mnemonic update the condition flags in the *cpsr*.

EXAMPLE 3.35 To help illustrate the advantage of conditional execution, we will take the simple C code fragment shown in this example and compare the assembler output using nonconditional and conditional instructions.

```
while (a!=b)
{
  if (a>b) a -= b; else b -= a;
}
```

Demo of Asm Program

prog_lec_2_2_2.ino and main_asm.S

- Program

```
AREA AddReg,CODE,READONLY ;Name this block of
code.
  ADR r0,ThumbProg +1 ;Generate branch target address
 ;and set bit 0,hence arrive
 ;at target in Thumb state.
 BX r0 ;Branch exchange to ThumbProg.
 CODE16 ;Subsequent instructions are Thumb code.
ThumbProg
 MOV r2,#2 ;Load r2 with value 2.
 MOV r3,#3 ;Load r3 with value 3.
 ADD r2,r3 ;r2 =r2 +r3
 ADR r0,ARMProg
 BX r0
 CODE32 ;Subsequent instructions are ARM code.
ARMProg
 MOV r4,#4
 MOV r5,#5
 ADD r4,r4,r5
 END
```

Summary

- Registers in Arm Cortex-M0+
- Assembly Syntax
  - **mov** and **bx** instructions
- Demo of **Asm** program

Ref: Ref3_ARM-Cortex-M0+ Technical Reference Manual.pdf