TI-RSLK

Texas Instruments Robotics System Learning Kit

Texas Instruments

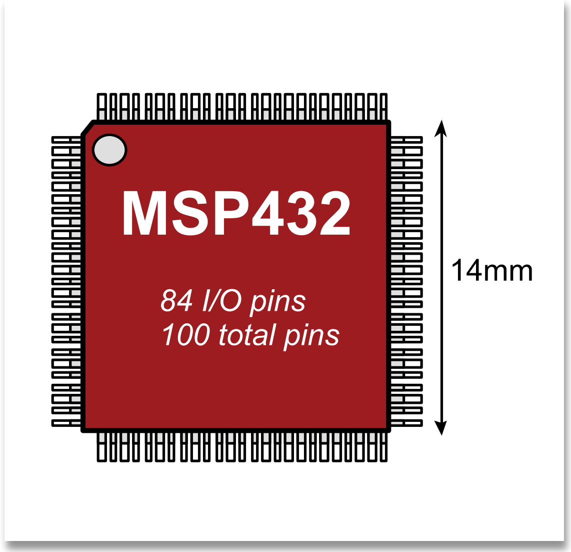
Module 3

Lecture: ARM Cortex M - Architecture

ARM Cortex M Architecture

You will learn in this module

- Cortex M Architecture
  - Buses
  - CISC versus RISC
  - Registers
  - Memory
  - Addressing modes

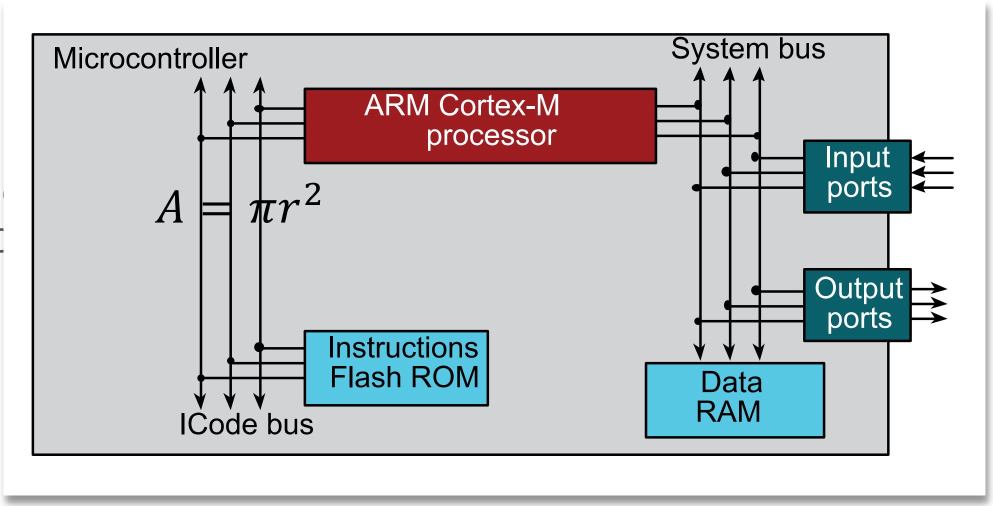


ARM Cortex M Architecture

ARM Cortex-M4 processor

- Harvard versus von Neumann architecture
- Different busses for instructions and data

- ICode bus - Fetch op codes from R
- System bus - Data from RAM and I/C
- Dcode bus - Debugging
- PPB bus - Private peripherals



$A = \pi r^2$

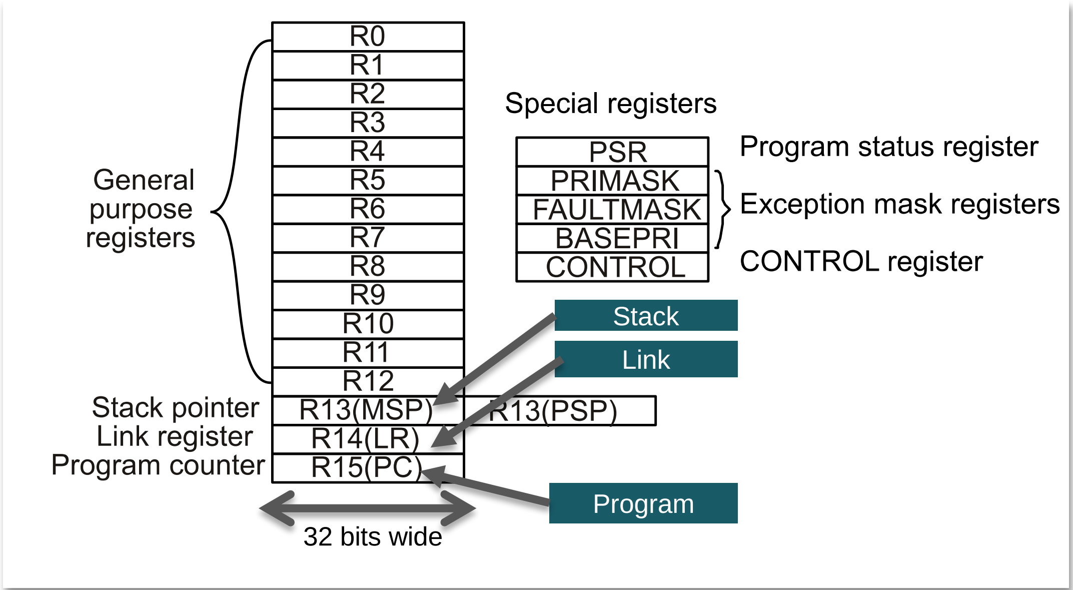
Reduced Instruction Set Computer (RISC)

CISC	RISC
Many instructions	Few instructions
Instructions have varying lengths	Instructions have fixed lengths
Instructions execute in varying times	Instructions execute in 1 or 2 bus cycles
Many instructions can access memory	Few instructions can access memory • Load from memory to a register • Store from register to memory
In one instruction, the processor can both • Read memory and • Write memory	• No one instruction can both read and write memory in the same instruction
Fewer and more specialized registers • Some registers contain data • Others contain addresses	• Many identical general purpose registers
Many different types of addressing modes	Limited number of addressing modes • Register, PC - relative • Immediate • Indexed

RISC machine

- Pipelining provides single cycle operation for many instructions
- Thumb-2 configuration employs both 16 and 32-bit instructions

Registers



Condition Code Bits		***Indicates***
N	negative	Result is negative
Z	zero	Result is zero
V	overflow	Signed overflow
C	carry	Unsigned overflow

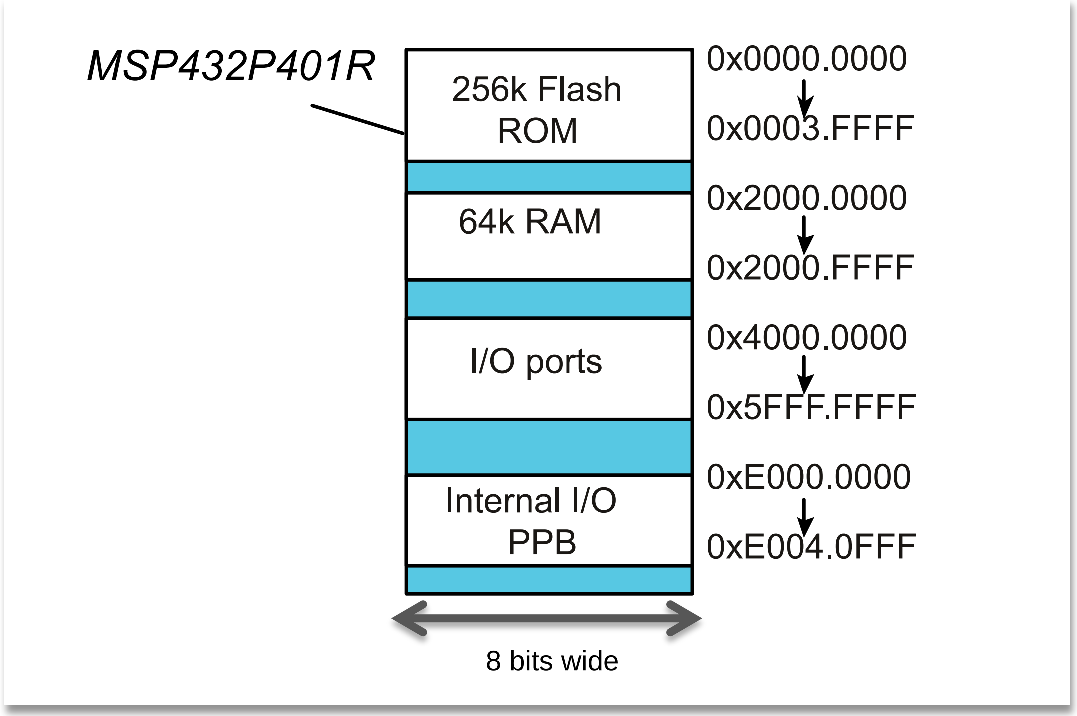
Where are data?

- Registers
- RAM
- ROM
- I/O ports

Where are commands?

- ROM (pointed to by PC)

Memory Map



For the detailed Memory Map go to http://www.ti.com/lit/ds/symlink/msp432p401r.pdf

Endianness

16-bit 1000 = 0x03E8

Address	Data
0x2000.0850	0x03
0x2000.0851	0xE8

Big Endian

Address	Data
0x2000.0850	0xE8
0x2000.0851	0x03

Little Endian

32-bit 1000000 = 0x000F4240

Address	Data
0x2000.0850	0x00
0x2000.0851	0x0F
0x2000.0852	0x42
0x2000.0853	0x40

Big Endian

Address	Data
0x2000.0850	0x40
0x2000.0851	0x42
0x2000.0852	0x0F
0x2000.0853	0x00

Little Endian

ASCII string “Jon” = 0x4A,0x6F,0x6E,0x00

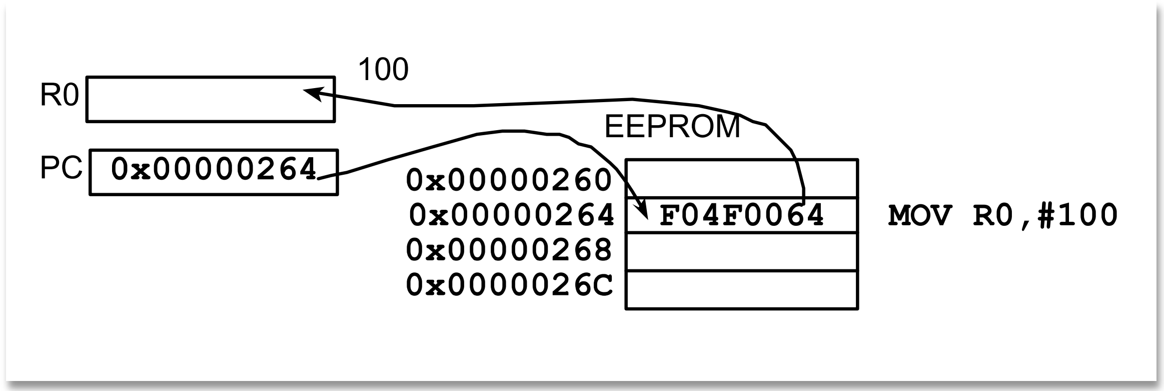
Address	Data
0x2000.0850	0x4A
0x2000.0851	0x6F
0x2000.0852	0x6E
0x2000.0853	0x00

Big Endian and Little Endian

Addressing Modes: immediate

- Register
- **Immediate**
- Indexed
- PC-relative

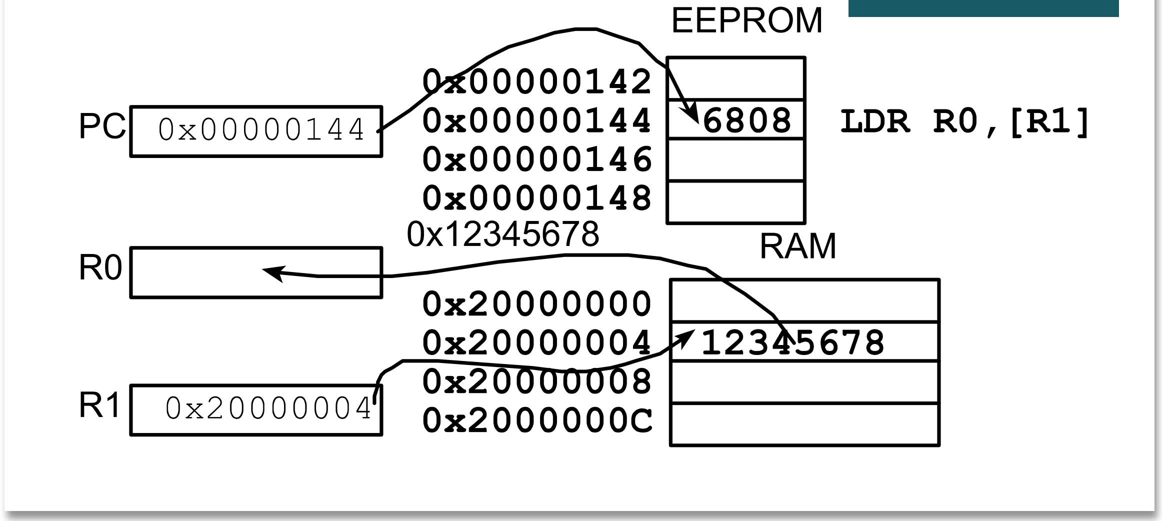
`MOV R0,#100`



Addressing Modes: Indexed

```
LDR R0,[R1]
```

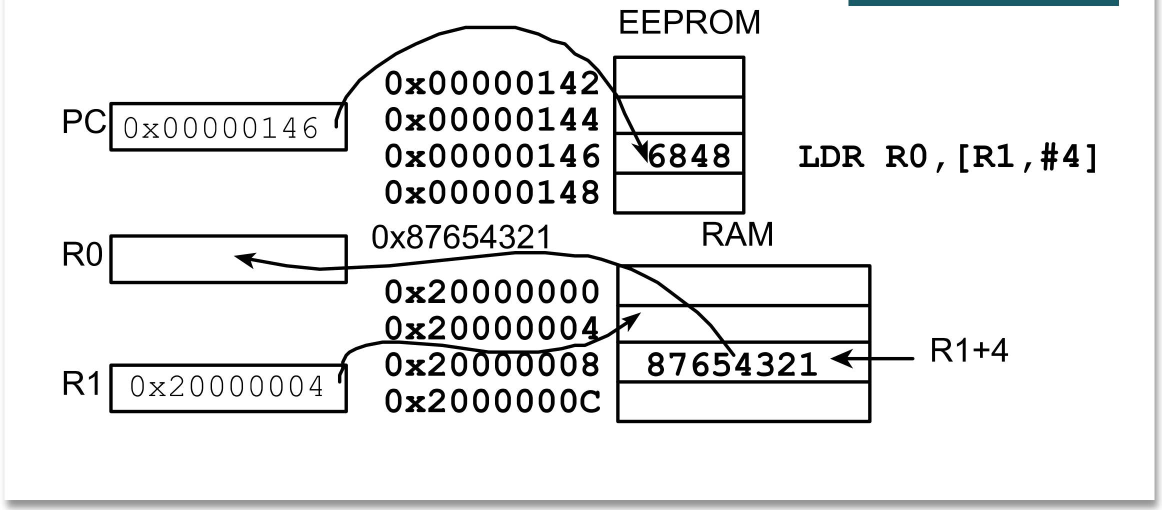
- Register
- Immediate
- **Indexed**
- PC-relative



Addressing Modes: Indexed

```
LDR R0,[R1,#4]
```

- Register
- Immediate
- **Indexed**
- PC-relative

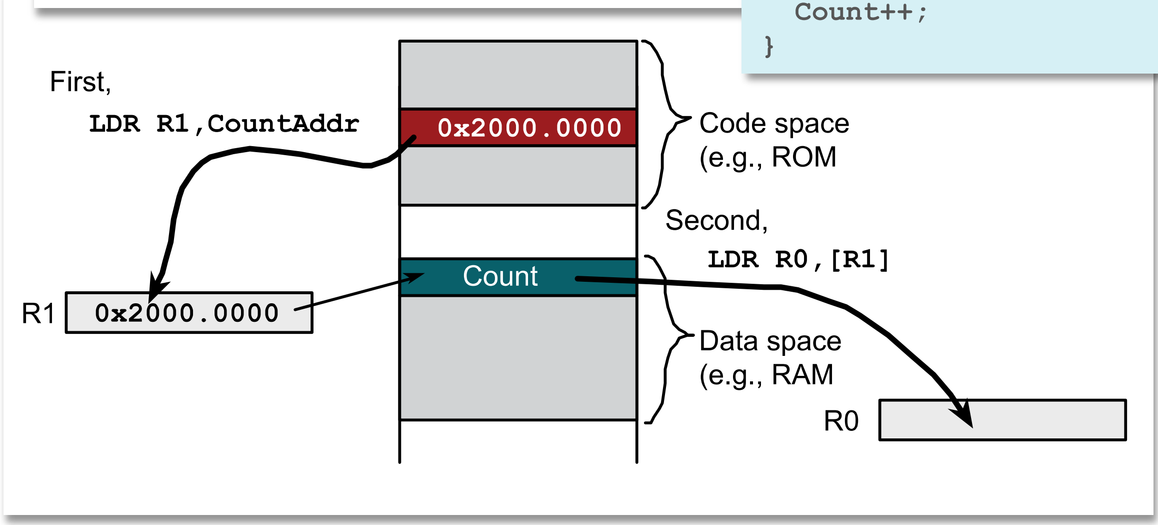


Variable Access: Load/store architecture

```
Increment:  .asmfunc
    LDR  R1,CountAddr  ; R1=pointer to Count
    LDR  R0,[R1]       ; R0=value of Count
    ADD  R0,R0,#1
    STR  R0,[R1]
    BX   LR
    .endasmfunc
CountAddr .field Count,32
```

- Register
- Immediate
- Indexed
- **PC-relative**

```
uint32_t Count;
void Increment(void){
  Count++;
}
```



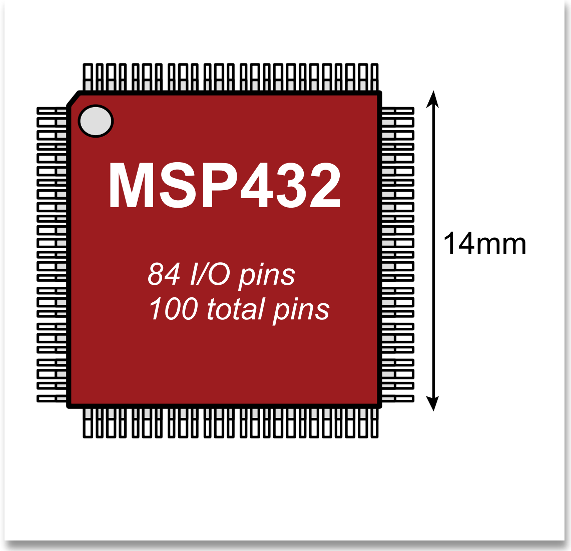
ARM Cortex M Architecture

Summary

- Architecture
  - Buses
  - Registers
  - Memory
  - Addressing modes

Terms:
- RISC vs CISC
- Little vs big endian
- Address vs data
- Variables

Register
Immediate
Indexed
PC-relative



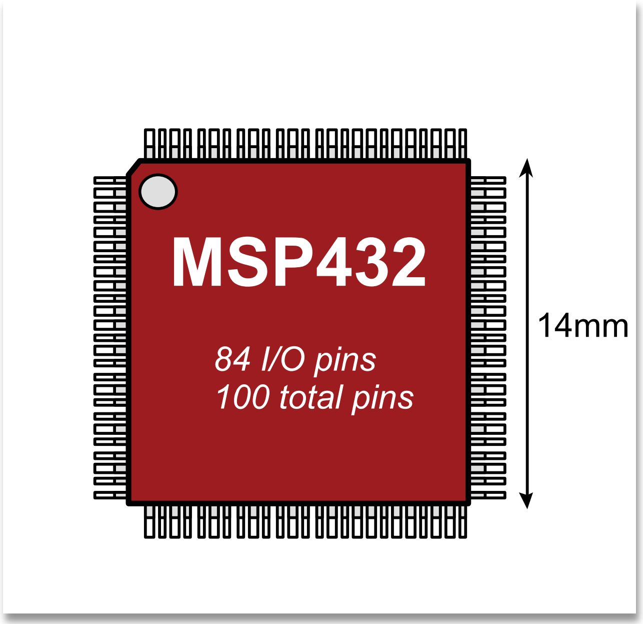
Module 3

Lecture: ARM Cortex M Assembly Programming

ARM Cortex M Assembly Programming

You will learn in this module

- Assembly Programming
  - Logical and shift operations
  - Addition, subtraction, multiplication and divide
  - Accessing memory
  - Stack
  - Functions, parameters
  - Conditionals
  - Loops



Logic Operations

A Rn	B Operand2	A&B AND	A|B ORR	A^B EOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

```
AND {Rd,} Rn, <op2> ;Rd=Rn&op2
ORR {Rd,} Rn, <op2> ;Rd=Rn|op2
EOR {Rd,} Rn, <op2> ;Rd=Rn^op2
```

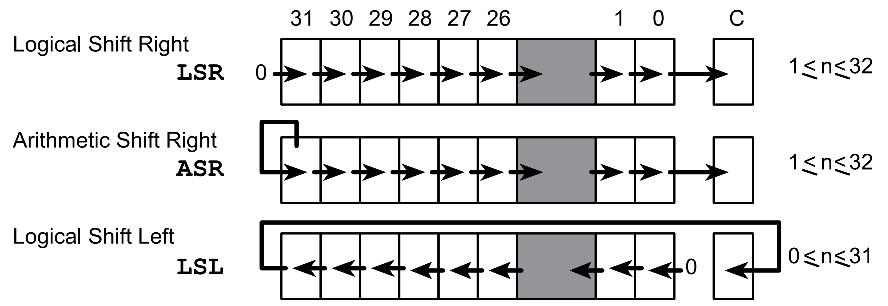
<op2>

- Register
- Register, shifted
- Constant

```
ORR R0,R1,R2
```

R1	0001	0010	0011	0100	0101	0110	0111	1000
R2	1000	0111	0110	0101	0100	0011	0010	0001
ORR	1001	0111	0111	0101	0101	0111	0111	1001

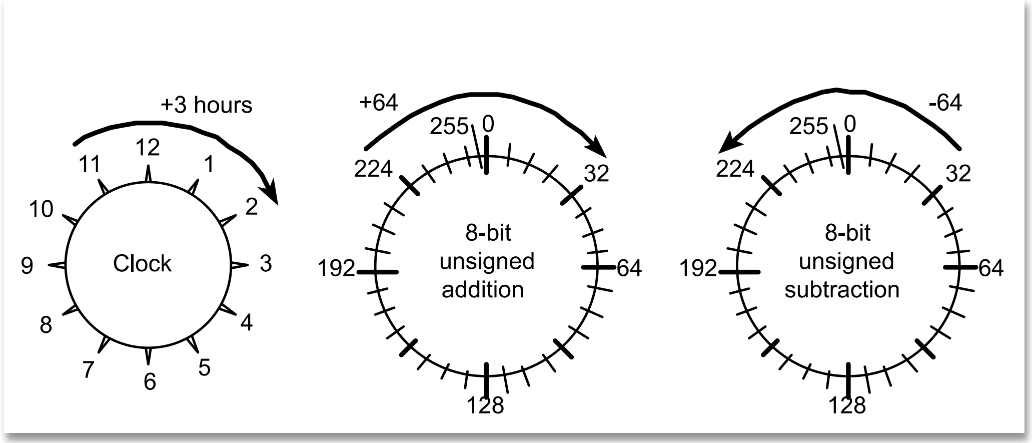
Shift Operations



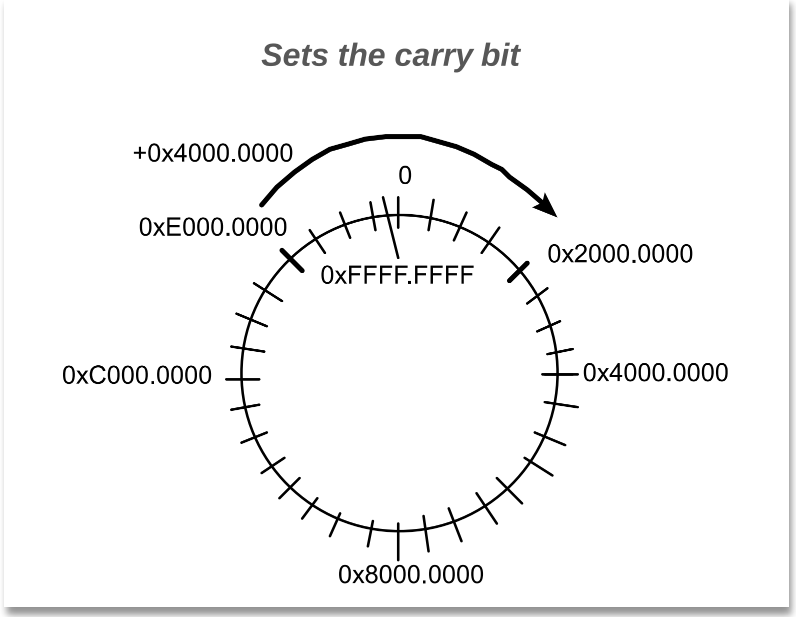
```
LSR Rd,   Rm,  Rs  ; logical shift right Rd=Rm>>Rs      (unsigned)
LSR Rd,   Rm,  #n  ; logical shift right Rd=Rm>>n       (unsigned)
ASR Rd,   Rm,  Rs  ; arithmetic shift right Rd=Rm>>Rs   (signed)
ASR Rd,   Rm,  #n  ; arithmetic shift right Rd=Rm>>n    (signed)
LSL Rd,   Rm,  Rs  ; shift left Rd=Rm<<Rs               (signed, unsigned)
LSL Rd,   Rm,  #n  ; shift left Rd=Rm<<n                (signed, unsigned)
```

Arithmetic Operations

- Addition/subtraction
  - Two n-bit → n+1 bits
- Multiplication
  - Two n-bit → 2n bits
- Avoid overflow
  - Restrict input values
  - Promote to higher, perform, check, demote
- Division
  - Avoid divide by 0
  - Watch for dropout
- Signed versus unsigned
  - Either signed or unsigned, not both
  - Be careful about converting types



Addition and Subtraction



Condition Code Bits		***Indicates***
N	negative	Result is negative
Z	zero	Result is zero
V	overflow	Signed overflow
C	carry	Unsigned overflow

<op2>

- Register
- Register, shifted
- Constant

```
ADD   {Rd,}  Rn,   <op2>    ;Rd = Rn + op2
SUB   {Rd,}  Rn,   <op2>    ;Rd = Rn - op2
CMP   Rn,          <op2>    ;Rn - op2
```

Multiplication and Division

```
MUL    {Rd,}  Rn,  Rm         ;Rd = Rn * Rm
UDIV   {Rd,}  Rn,  Rm         ;Rd = Rn/Rm unsigned
SDIV   {Rd,}  Rn,  Rm         ;Rd = Rn/Rm signed
```

```
uint32_t N,M;
// times 0.6
void Fun(void){
  M = 3*N/5;
}
```

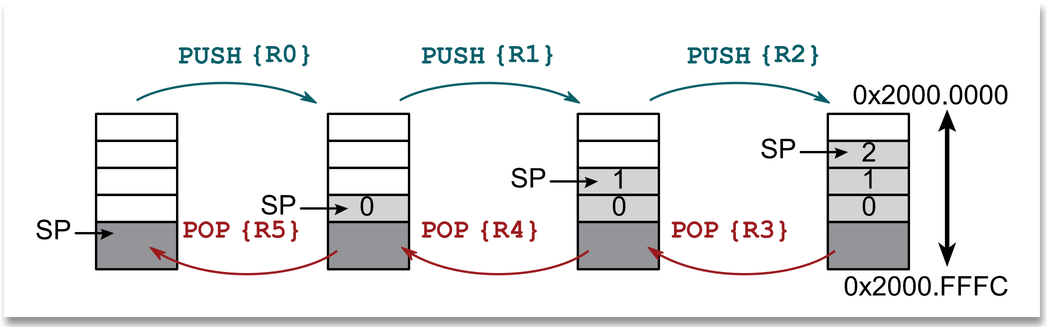
```
      .data
      .align  2
N     .space  4
M     .space  4
      .text
      .align  2
Fun:   .asmfunc
   LDR  R3, Naddr    ; R3 = &N (R3 points to N)
   LDR  R1, [R3]     ; R1 = N
   MOV  R0, #3       ; R0 = 3
   MUL  R1, R0, R1   ; R1 = 3*N
   MOV  R0, #5       ; R0 = 5
   UDIV R0, R1, R0   ; R0 = 3*N/5
   LDR  R2, MAddr    ; R2 = &M (R2 points to M)
   STR  R0, [R2]     ; M = 3*N/5
   BX   LR
      .endasmfunc
NAddr .field N,32
MAddr .field M,32
```

Stack

```
PUSH {R0}
PUSH {R1}
PUSH {R2}
POP  {R3}
POP  {R4}
POP  {R5}
```

Push
1. SP=SP-4
2. Store at SP

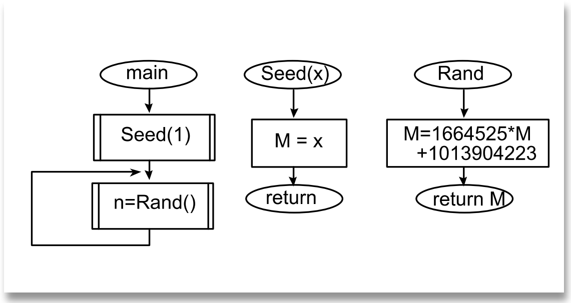
POP
1. Read at SP
2. SP=SP+4



Usage
- Temporary storage
- Local variables

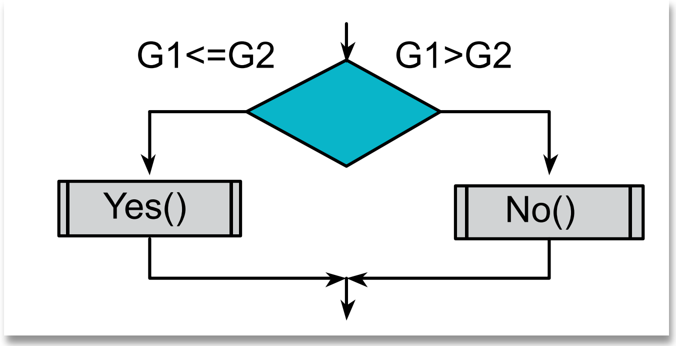
Function calls

```
        .data
        .align  2
M       .space  4
        .text
        .align  2
Seed:   .asmfunc
        LDR R1,MAddr ; R1=&M
        STR R0,[R1]  ; set M
        BX  LR
        .endasmfunc
Rand:   .asmfunc
        LDR R2,MAddr ; R2=&M, address of M
        LDR R0,[R2]  ; R0=M, value of M
        LDR R1,Slope
        MUL R0,R0,R1 ; R0 = 1664525*M
        LDR R1,Offst
        ADD R0,R0,R1 ; 1664525*M+1013904223
        STR R0,[R2]  ; store M
        LSR R0,#24   ; 0 to 255
        BX  LR
        .endasmfunc
MAddr   .field M,32
Slope   .field 1664525,32
Offst   .field 1013904223,32
```



```
        .data
        .align  2
n       .space  4
        .text
        .align  2
main:   .asmfunc
        MOV R0,#1
        BL  Seed
loop    BL  Rand
        LDR R1,nAddr
        STR R0,[R1]
        B   loop
        .endasmfunc
nAddr   .field n,32
```

Conditionals



```
        LDR R3,G2Addr ; R3=&G2, address of G2
        LDR R2,[R3]   ; R2=G2,  value of G2
        LDR R0,G1Addr ; R0=&G1, address of G1
        LDR R1,[R0]   ; R1=G1,  value of G1
        CMP R1,R2     ; compare G1 G2
        BHI isNo
isYes   BL  Yes       ; G1<=G2
        B   done
isNo    BL  No
done

G1Addr .field G1,32
G2Addr .field G2,32
```

Instruction	Branch if
B target	; always
BEQ target	; equal (signed or unsigned)
BNE target	; not equal (signed or unsigned)
BLO target	; unsigned less than
BLS target	; unsigned less than or equal to
BHS target	; unsigned greater than or equal to
BHI target	; unsigned greater than
BLT target	; signed less than
BGE target	; signed greater than or equal to
BGT target	; signed greater than
BLE target	; signed less than or equal to

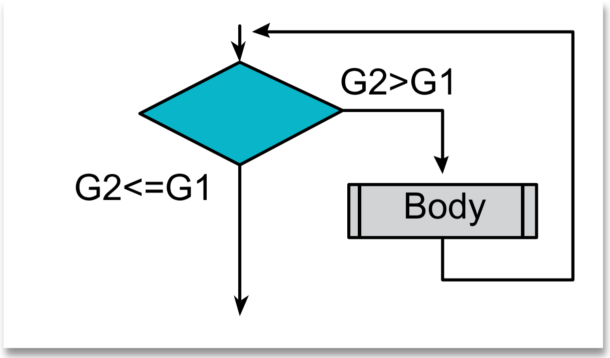
```
if(G2<=G1){
  Yes();
}else{
  No();
}
```

Think of the three steps

1) bring first value into a register,
2) compare to second value,
3) conditional branch, bxx

(where xx is eq ne lo ls hi hs gt ge lt or le).
The branch will occur if (first is xx second).

While Loops

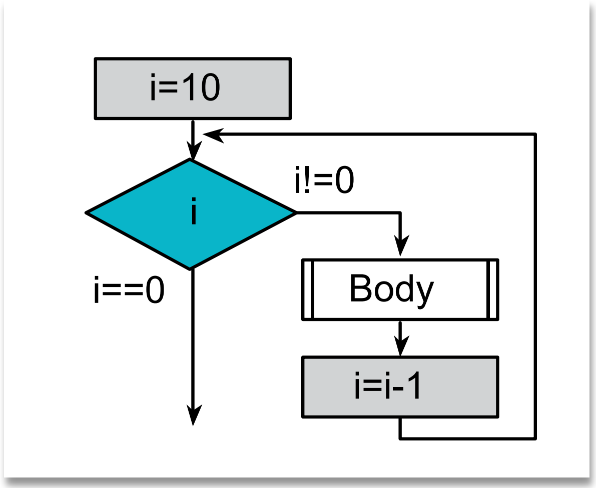


```
       LDR R3,G2Addr ; R3=&G2, address of G2
       LDR R2,[R3]   ; R2=G2,  value of G2
       LDR R0,G1Addr ; R0=&G1, address of G1
       LDR R1,[R0]   ; R1=G1,  value of G1
loop   CMP R1,R2     ; compare G1 G2
       BLS done
       BL  Body      ; G1>G2
       B   loop
done

G1Addr .field G1,32 ;unsigned 32-bit number
G2Addr .field G2,32 ;unsigned 32-bit number
```

```
while(G2>G1){
  Body();
}
```

For Loops



```
for(i=10; i!=0; i--){
  Body(); // 10 times
}
```

```
      MOV  R4,#10
loop  CMP  R4,#0
      BEQ  done
      BL   Body
      SUB  R4,R4,#1
      B    loop
done
```

```
      MOV  R4,#10
loop  BL   Body
      SUBS R4,R4,#1
      BNE  loop
```

ARM Cortex M Assembly Programming

Summary

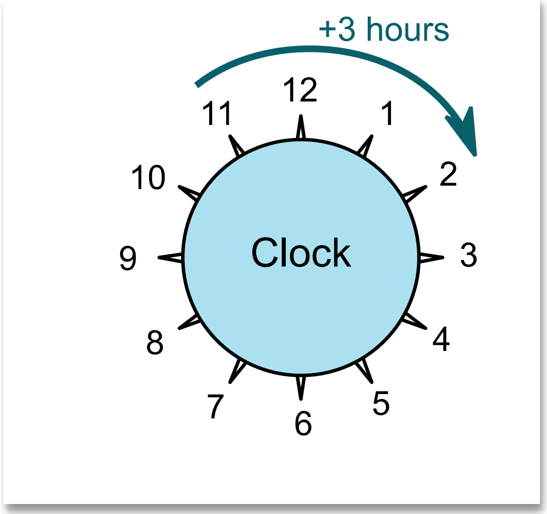
- Programming
  - Accessing memory
  - Logical and shift operations
  - Addition, subtraction, multiplication and divide
  - Stack
  - Functions, parameters
  - Conditionals
  - Loops

Register

Immediate

Indexed

PC-relative


+3 hours
Clock
12
1
2
3
4
5
6
7
8
9
10
11