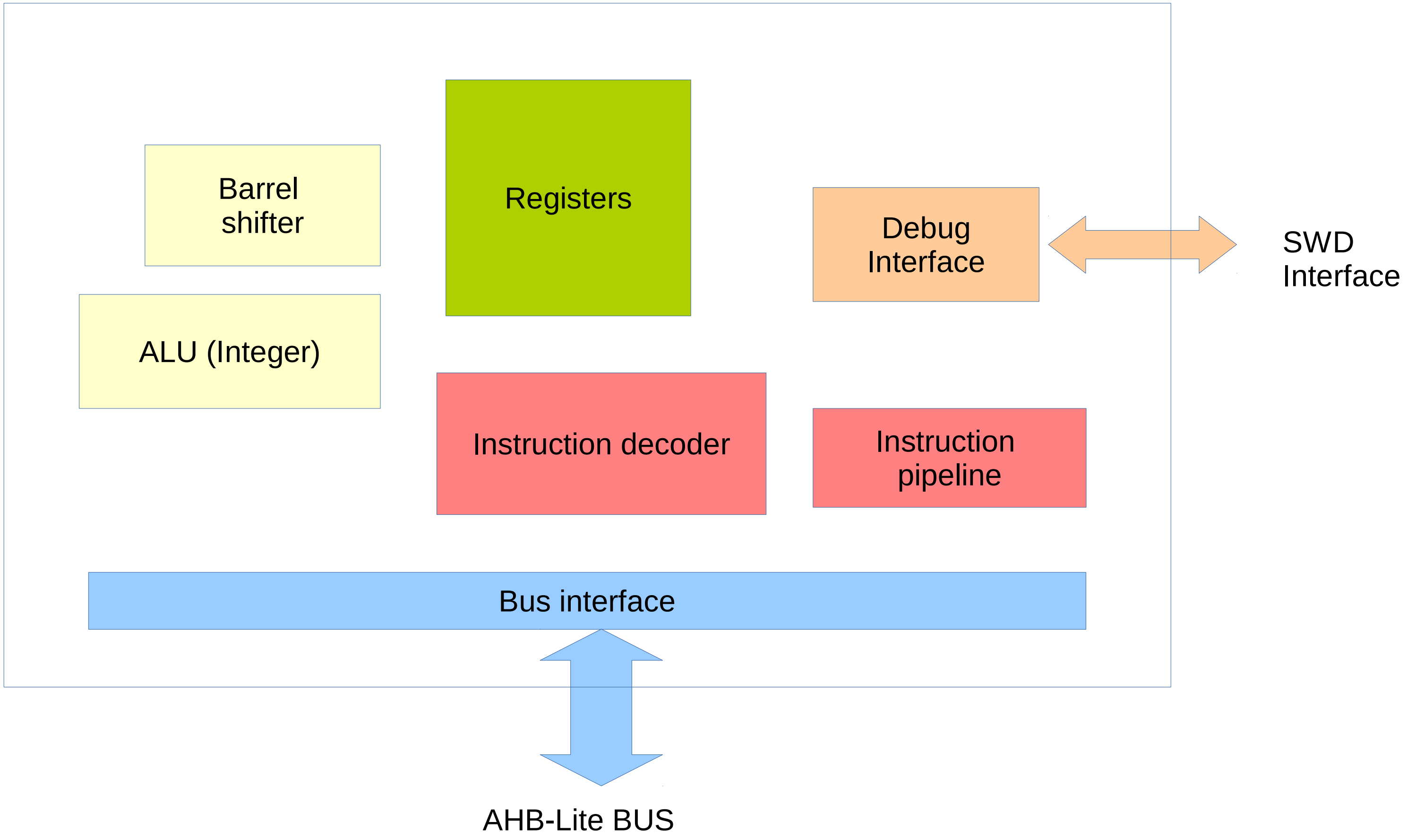
ARM Assembler

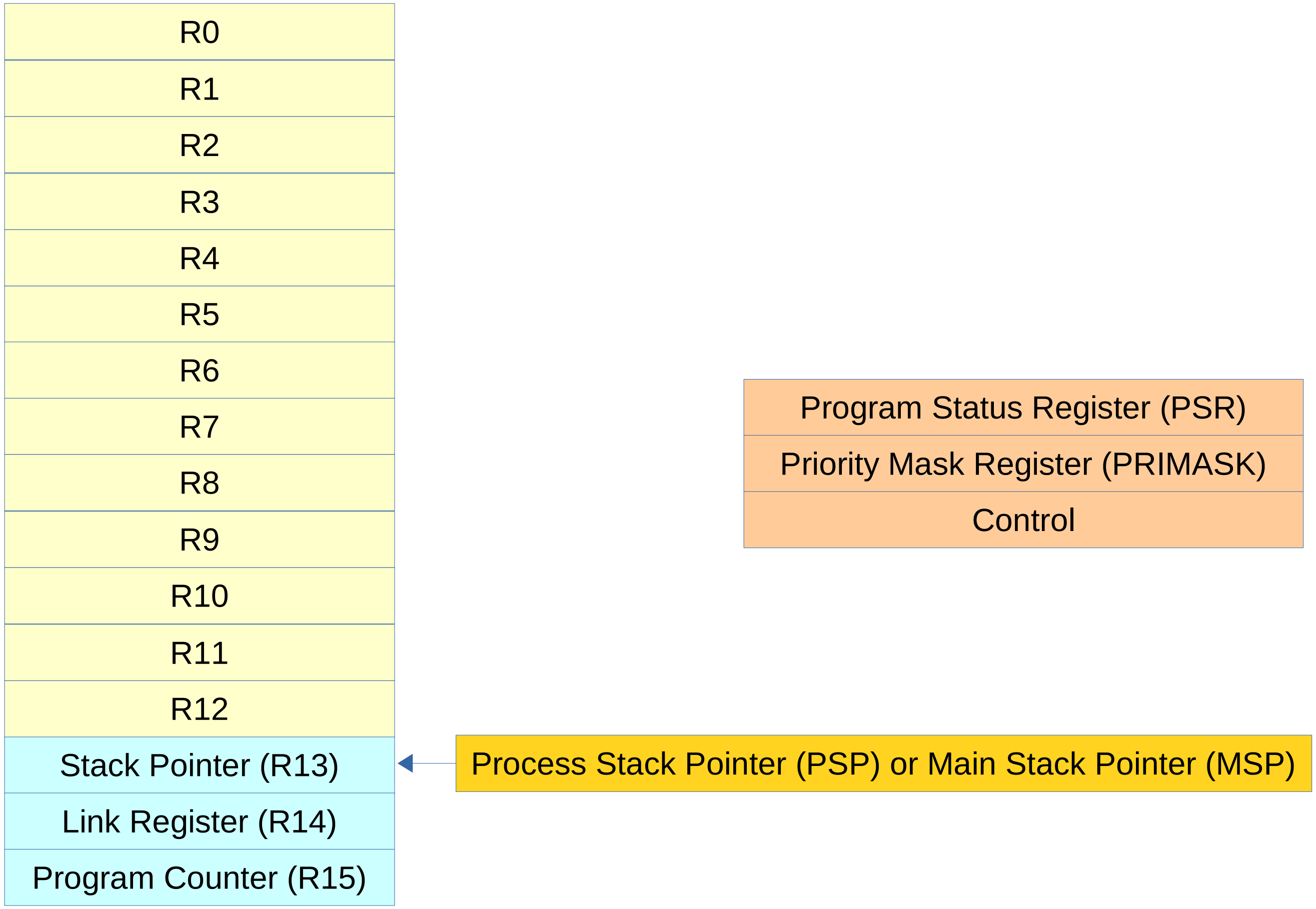
ARM Cortex microcontrollers

Cortex M0 Core



ARM Cortex microcontrollers

Cortex M0 Registers



R0

R1

R2

R3

R4

R5

R6

R7

R8

R9

R10

R11

R12

Stack Pointer (R13)

Link Register (R14)

Program Counter (R15)

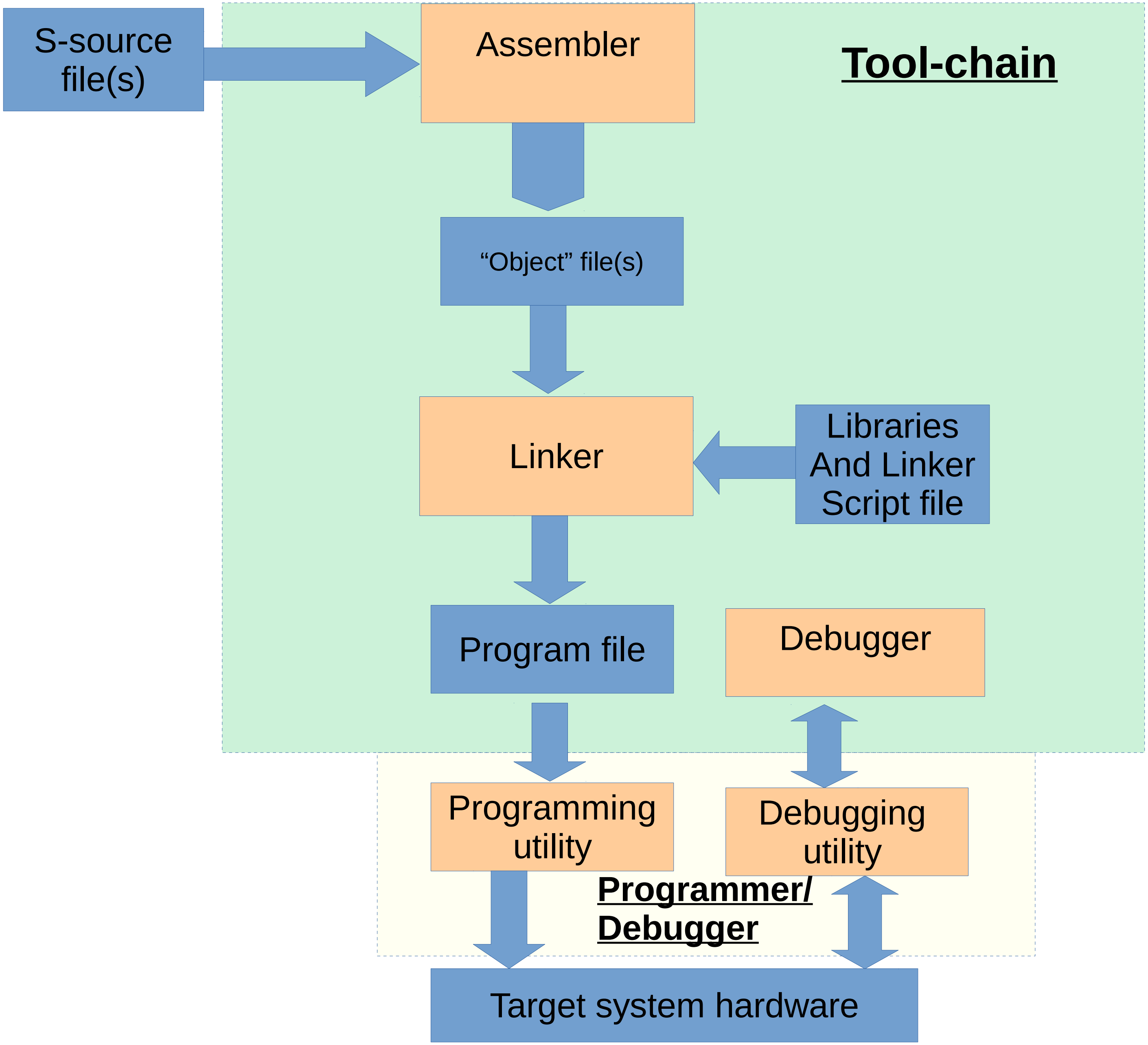
Process Stack Pointer (PSP) or Main Stack Pointer (MSP)

Program Status Register (PSR)

Priority Mask Register (PRIMASK)

Control

Programming



ARM Assembler

- Microprocessor instruction decoders understand numeric instructions
- Humans don't
- Assembly language make machine code (slightly) more readable

ARM Assembler

- Assembler is not used much these days:
  - Not portable
  - Difficult to read and maintain
  - Not necessarily faster than C
- Why do this so?
  - It improves our understanding of how computers work
  - Sometimes assembler is needed for time critical or severely constrained systems

ARM Assembler

LDR	Rt, [Rn, *<Rm|#imm>]*	Load Register with word	-	*Memory access instructions*

e.g.

```
LDR R0,=1234
LDR R0,[R1]
```

ARM Assembler

`ADD{S}`	*{Rd,}* Rn*, <Rm*|*#imm>*	Add	N,Z,C ,V	*Arithmetic instructions*

e.g.

```
    ADDS R0,R0,#1
    ADDS R0,R0,R1
```

ARM Assembler

STR	Rt, [Rn, *<Rm|#imm>]*	Store Register as word	-	*Memory access instructions*

e.g.

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

ARM Assembler

Not everything is possible: addressing modes

ARM Assembler

```
        ; This is a comment
        ; Symbols start up against the left hand margin
        ; The following symbols reside in RAM
        AREA DATA

Result DCD 0            ; allocate space for the result of a calculion
Stack  SPACE 0x100  ; allocate some space for the stack

        ; The following symols are in the CODE section (ROM,Executable, readonly)

        AREA THUMB,CODE,READONLY
        ; EXPORTED Symbols can be linked against
                EXPORT Reset_Handler
                EXPORT __Vectors
        ; Minimal interrupt vector table follows
        ; First entry is initial stack pointer (end of stack)
        ; second entry is the address of the reset handler
__Vectors
                DCD Stack+0x100
                DCD Reset_Handler
        ; Define some constant operands
Num1   DCD 0x7fffffff
Num2   DCD 2
```

ARM Assembler

```
        ; 'Main' program goes here
Reset_Handler
                LDR R1,=Num1
                LDR R2,=Num2
                LDR R0,[R1]
                LDR R1,[R2]
                ADDS R0,R0,R1
                LDR R1,=Result
                STR R0,[R1]
stop    B stop
        end
```

ARM Assembler

Demo on Keil

ARM Assembler

```
// global variables
int X;
int Y;

Y=0;
for (X=0;X<10;X++)
       Y = Y + X;
```

ARM Assembler

```
// global variable
X DCD 0
Y DCD 0
:
Y=0;                LDR R0,=0
                    LDR R1,=Y
                    STR R0,[R1]


;;for (X=0;X<10;X++)
(X=0)
                    LDR R0,=0
                    LDR R1,=X
                    STR R0,[R1]
;Y=Y+X
Loop_Start:
                    LDR R1,=Y
                    LDR R2,[R1]
                    ADDS R2,R2,R0
                    STR R2,[R1]
```

ARM Assembler

```
; (X++)
                LDR R1,=X
                LDR R0,[R1]
                ADDS R0,R0,#1
                STR R0,[R1]

; (X<10)
                LDR R1,=X
                LDR R0,[R1]
                CMP R0,#10
                BLT Loop_Start
```