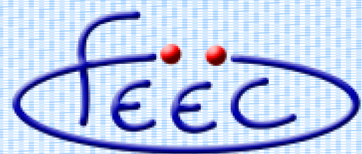


EA871

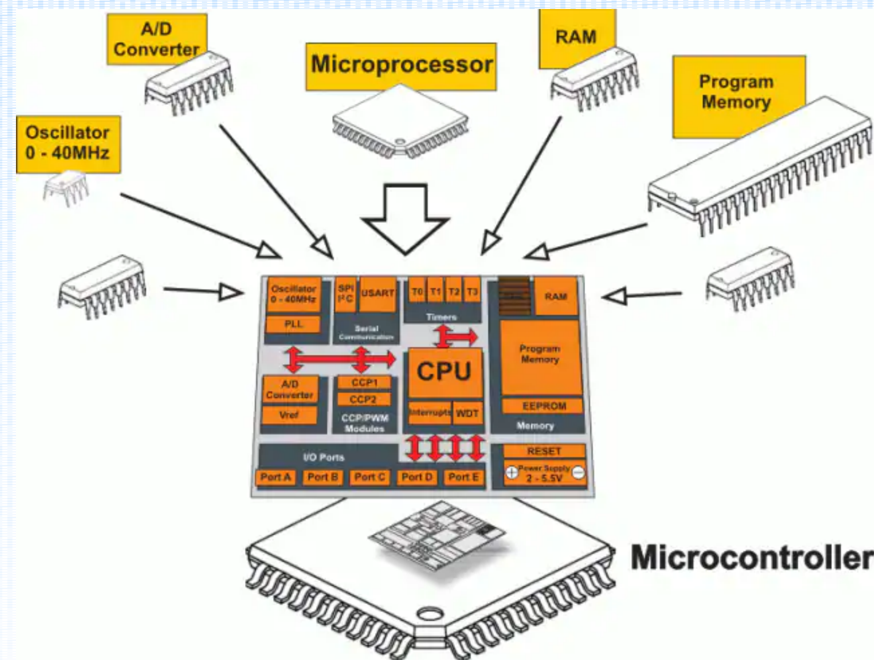
Linguagem *Assembly*

ARM Thumb e GNU *Assembler* (GAS)

Wu Shin – Ting
DCA – FEEC - Unicamp
Segundo Semestre de 2020

Conceitos

- **Microcontrolador**: é um sistema computacional, incluindo processador, sistema de memória e módulos de interface com periféricos, integrado numa única pastilha (*system on chip* ou SoC). É de baixo custo, de baixo consumo de energia e resistente a condições adversas.



Fonte: https://www.arrow.com/en/research-and-events/articles/engineering-basics-what-is-a-microcontroller

Conceitos

- **Arquitetura de um microcontrolador**: é a forma como os componentes são conectados para constituí-lo.
- Classificação quanto à transferência de dados entre o sistema de memória e o **processador**:
  - **Arquitetura von Neumann**: as instruções e os dados compartilham o mesmo espaço de memória e mesmo barramento. Processador não consegue ler instruções (da memória) e escrever dados (na memória) simultaneamente.
  - **Arquitetura Harvard**: as instruções e os dados são acessados por barramentos distintos. Pode-se paralelizar o acesso de leitura de instruções e o acesso de escrita de dados.

Conceitos

- Classificação quanto à transferência de dados entre os periféricos e o **processador**:
  - **E/S mapeada em memória**: os registradores (controle, estado e de dados) dos periféricos e as unidades de memória são mapeados no mesmo espaço de endereços, fazendo uso de mesmo conjunto de instruções para acessos de leitura e de escrita.
  - **E/S isolada**: o espaço de endereços do sistema de memória é diferente do espaço de endereços dos periféricos, fazendo uso de instruções distintas para acessos de leitura e escrita.

Conceitos

- **Processador**: é um circuito capaz de transformar as **instruções**, armazenadas no sistema de memória, em sinais de controle que coordenam as operações do restante dos componentes do sistema, a fim de executar tarefas programadas.
- **Níveis de abstração de um processador**:
  - Circuitos combinacionais e sequenciais.
  - Transferências de sinais entre os registradores (*Register Transfer Level*) na base dos sinais de relógio.
  - Blocos funcionais, como unidade lógico-aritmética, unidade de controle, sistema de memória, banco de registradores, e suas interconexões.
  - Arquitetura de repertório/conjunto de instruções (ISA).

Conceitos

- **Arquitetura do Conjunto/Jogo de Instruções** (*Instruction Set Architecture*, **ISA**): é uma abstração da organização dos blocos funcionais de um processador por meio de um "vocabulário" de **instruções**.

- **Instrução**: é uma forma de **codificação** de uma operação que um processador consegue executar. Tipicamente, contém o **código de operação** e os **operandos** necessários.

Código de Operação	Operandos

Conceitos

- Classificação da arquitetura quanto à complexidade das instruções
  - **CISC** (*Complex Instruction Set Computer*): arquitetura com um conjunto complexo de instruções, de tamanho e de tempo de execução bem variados. Exemplo: processadores x86.
  - **RISC** (*Reduced Instruction Set Computer*): arquitetura com um número reduzido de instruções, procurando manter constantes o tamanho das instruções e o seu tempo de execução. Exemplo: ARM (***A****dvanced* ***R****ISC* ***M****achine*) e MIPS (***M****icroprocessor without* ***I****nterlocked* ***P****ipeline* ***S****tages*).

Conceitos

- Classificação de arquitetura quanto à forma de armazenamento dos operandos:
  - **Organização por pilha**, **instruções de zero endereço**: operandos sempre no topo de uma pilha (memória).
  - **Organização por acumulador**, **instruções de um endereço**: um dos operandos sempre no registrador acumulador.
  - **Organização por registradores**, **instruções de dois ou três endereços**
    - **Registrador-registrador**: operandos armazenados nos registradores. É conhecida como **arquitetura *load-store***.
    - **Registrador-memória**: um operando armazenado no registrador e outro na memória.
    - **Memória-memória**: operandos armazenados no sistema de memória.

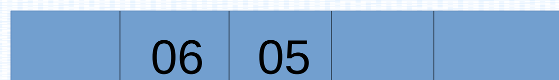
Conceitos

- **Modos de endereçamento**: especificam como os *bits* no campo dos operandos devem ser interpretados e processados para obter os **endereços efetivos** dos operandos.
- **Endereço efetivo**: é a posição real de um operando. Um operando pode estar codificado na instrução, estar localizado num registrador ou na memória.
- **Tipos de dados**: tamanho dos operandos, em *bytes* (1, 2, 4, 8).
- **Formatos de dados na memória**: ordenação dos *bytes* de um dado na memória.

  Little-endian: de *byte* menos significativo para mais significativo.

  Big-endian: de *byte* mais significativo para menos significativo.

$1268_d \rightarrow 05_06_h$



Little-endian

Big-endian

Um Modelo de Programação

- **Repertório de instruções**
- **Espaços de endereços**: locais onde estão armazenadas as instruções e os operandos.
- **Pilhas:** locais para armazenamento de dados temporários
- **Banco de registradores** de propósito geral
- **Registradores de função especial**: ponteiros de pilhas SP, contador de programa PC, registrador de retorno LR.
- **Registradores de estado**: registradores que guardam os *bits* de condição N (negativo), Z (zero), C (*carry*), V (*overflow*) após operações lógico-aritméticas.

Linguagem de Programação

- **Código de máquina**: **codificação binária** de instruções, num formato interpretável pelas máquinas.
  - acesso direto às características particulares do *hardware* → melhor controle do uso de *hardware* → melhores resultados em tempo de execução e em tamanho de código.
  - extenso, propenso a erros e pouco inteligível.

- **Linguagem de montagem** ou **simbólica** (*assembly*): **codificação mnemônica** de instruções, num formato mais legível para humanos.
  - disponivel em todos os processadores.
  - altamente dependente da organização do processador.

Instruções

```
00000810:  cmp r4,#2
 51          beq subt
00000812:  beq (AsmSection)+0x42 (0x842); 0x0
 53          cmp r4,#3
00000814:  cmp r4,#3
 54          beq mult
00000816:  beq (AsmSection)+0x4a (0x84a); 0x0
 56          cmp r4,#4
00000818:  cmp r4,#4
 57          beq land
0000081a:  beq (AsmSection)+0x52 (0x852); 0x0
 59          cmp r4,#5
0000081c:  cmp r4,#5
 60          beq lor
0000081e:  beq (AsmSection)+0x5a (0x85a); 0x0
 62          cmp r4,#6
00000820:  cmp r4,#6
 63          beq lxor
00000822:  beq (AsmSection)+0x62 (0x862); 0x0
 65          cmp r4,#7
00000824:  cmp r4,#7
 66          beq llsl
00000826:  beq (AsmSection)+0x6a (0x86a); 0x0
```

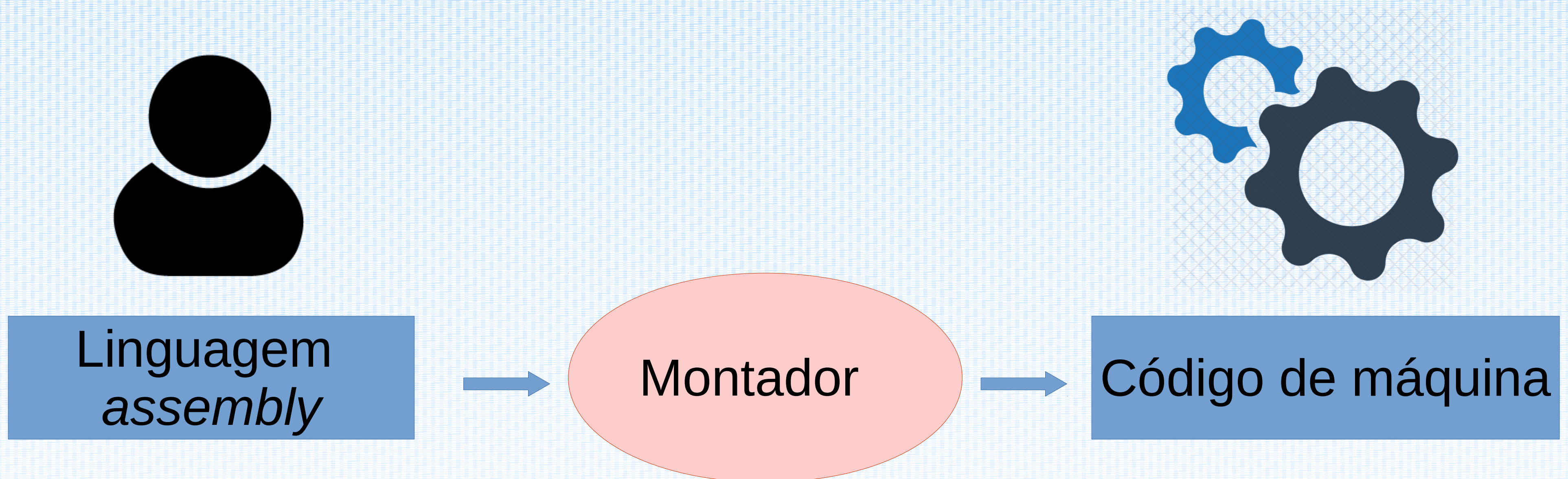
assembly

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000800	1F	B5	23	48	03	68	06	22	02	21	01	2C	FD	DB	14	D0
00000810	02	2C	16	D0	03	2C	18	D0	04	2C	1A	D0	05	2C	1C	D0
00000820	06	2C	1E	D0	07	2C	20	D0	08	2C	22	D0	09	2C	24	D0
00000830	0A	2C	26	D0	14	2C	28	D0	E7	E7	13	1C	5B	18	03	60
00000840	E3	E7	13	1C	5B	1A	43	60	DF	E7	13	1C	4B	43	83	60
00000850	DB	E7	13	1C	0B	40	C3	60	D7	E7	13	1C	0B	43	03	61
00000860	D3	E7	13	1C	4B	40	43	61	CF	E7	13	1C	8B	40	83	61
00000870	CB	E7	13	1C	CB	40	C3	61	C7	E7	13	1C	CB	41	03	62
00000880	C3	E7	13	1C	0B	41	43	62	BF	E7	1F	BD	C0	46	C0	46
00000890	00	F0	FF	1F	C0	46	C0	46	C0	46	C0	46	C0	46	C0	46
000008A0	80	B5	00	AF	00	BE	BD	46	80	BD	C0	46	80	B5	00	AF
000008B0	05	4A	06	49	06	4B	D1	50	06	4A	88	23	5B	01	00	21
000008C0	D1	50	BD	46	80	BD	C0	46	00	E0	00	E0	00	00	00	00
000008D0	08	0D	00	00	00	70	04	40	80	B5	00	AF	05	4A	06	4B
000008E0	D3	1A	05	4A	10	1C	00	21	1A	1C	00	F0	41	F8	BD	46
000008F0	80	BD	C0	46	1C	F0	FF	1F	00	F0	FF	1F	00	F0	CA	F8
00000900	FF	F7	D4	FF	FF	F7	E8	FF	00	F0	F8	F8	00	F0	76	F8

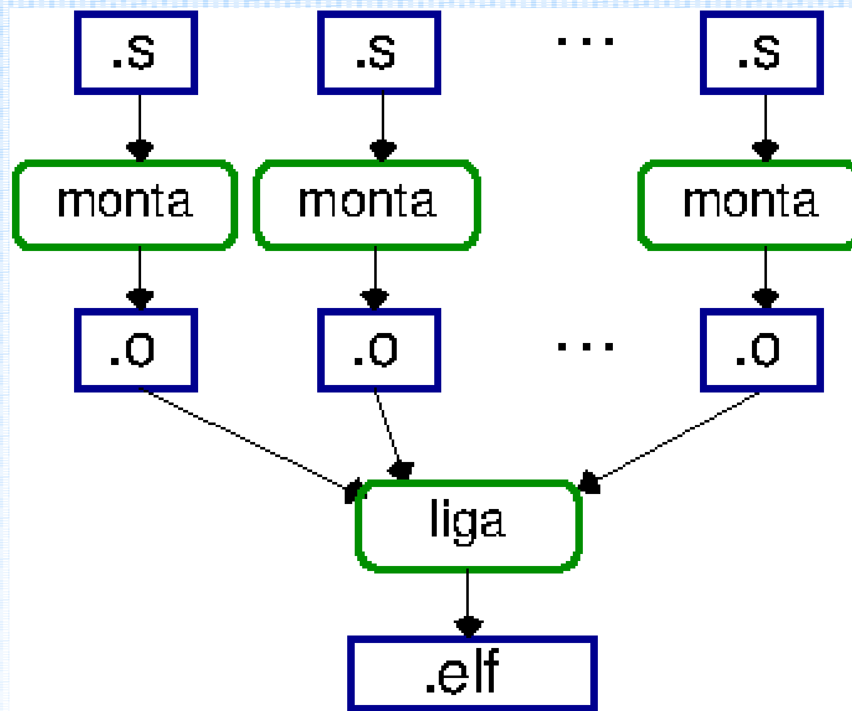
Código de máquina em hexadecima
(formato binário na máquina!)

Conceitos

- **Montador** (*assembler*): é um aplicativo que traduz a linguagem simbólica (*assembly*) em código de máquina.



Montador



Código-fonte em linguagem simbólica

Código de máquina

Código executável

Montador GNU

- ***GNU Assembler*, GAS**: é o montador padrão do Projeto GNU. Suporta uma variedade de linguagens *assembly* em diferentes arquiteturas.
- Formato de cada linha de declaração

  [<rótulo>:] [<instrução ou diretiva>} @ comentário

  - rótulo: corresponde ao endereço da instrução
  - instrução: é traduzida para códigos binários de máquina
  - diretiva: complementa as informações necessárias para a montagem de um programa; não é traduzida para códigos binários.
  - comentários

É diferente do montador ARM (*ARM assembler*)!

Algumas diretivas do montador GNU

.text	Instruções ou dados inicializados depois dessa declaração serão colocados no segmento de códigos.
.data	dados inicializados depois dessa declaração serão colocados no segmento de dados.
.bss	dados (não inicializados) depois dessa declaração serão colocados no segmento de dados
.global <símbolo>	torna o símbolo visível ao ligador para ligar com os diferentes trechos de códigos.
.section <sect>{,”flags”}	define um segmento de instruções, dados inicializados ou não-inicializados. Seção .rodata é um segmento só de acesso de leitura (*read only*)
.equ <símbolo>, <valor>	seta o valor ao símbolo
.word <word1>{,<word2>}	insere lista de valores de 32 *bits*
.hword <shirt1> {,<short2>}	insere lista de valores de 16 *bits*
.align n	alinhe os endereços em múltiplos de 2^n
.type	define o tipo de um símbolo
.end	fim de um arquivo (opcional)

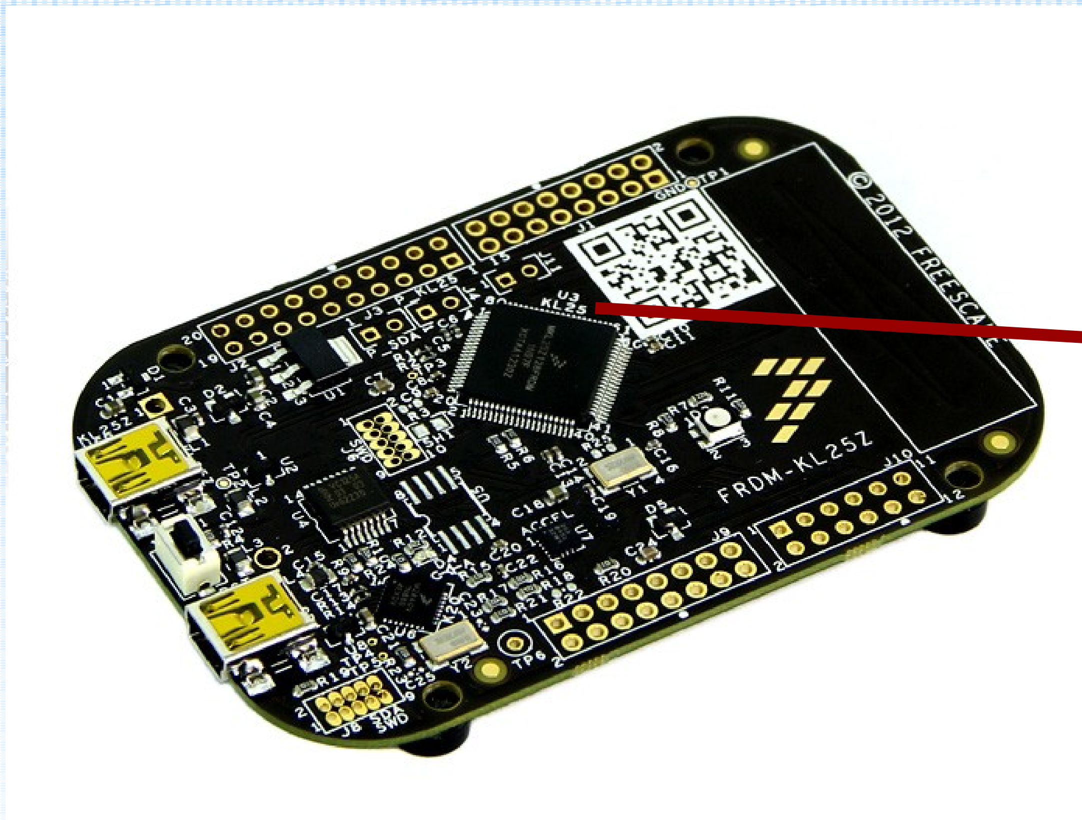
Um Exemplo

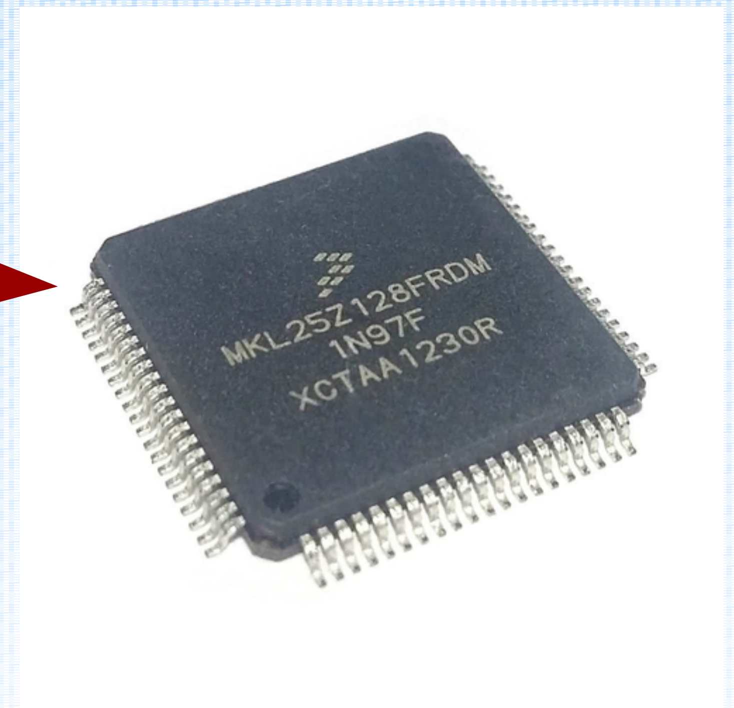
```
    .text                  @monta um segmento de instruções e dados
    .align 1               @alinha os códigos binários em end múltiplos de 2 bytes
    .global main           @<main> é um símbolo global usado pelo ligador
    .type main function    @define símbolo <main> como nome de uma função
```

Instruções de máquina

```
    .data                  @monta um segmento de dados
    .align 2               @alinha os códigos binários em end múltiplos de 4 bytes
varC:                      @rótulo
    .word 0x1ffff000       @valor do rótulo em 4 bytes
    .end                   @marca o fim de um arquivo em assembly
```

FRDMKL25Z





Microcontrolador Kinetis L Série KL25

Microcontrolador
Kinetis L Séries KL2x

Core

- **ARM® Cortex®-M0+ core up to 72 MHz** (up to 96 MHz for high-speed run)
- Single Wire Debug
- Interrupt Controller
- Micro Trace Buffer

System

- Power Management Controller
- Bit Manipulation Engine
- DMA
- Internal Watchdogs
- Low-Leakage Wakeup

Memory

- **Program Flash** 32–512 KB
- **SRAM** 4–128 KB
- 32-Byte Register File
- **BOOTROM** 16–32 KB

Clocks

- High Accuracy 60 M/48 MHz IRC
- Low-Speed IRC 8 M/2 MHz or 4 M/32 kHz
- PLL/FLL
- Low-/High-Frequency Oscillators
- 1 kHz LPO

Security and Integrity

- Cyclic Redundancy Check (CRC)
- Flash Access Controller
- MMCAU
- True Random Number Generator
- Unique ID

Analog

- 16-bit SAR ADC
- 2 x High-Speed ACMP
- 12-bit DAC
- V_{REF}

Timers

- 1 x 6-ch. Timer/PWM
- 2 x 2-ch. Timer/PWM
- 2 x Low-Power Timer
- Independent RCT
- 4-ch. Periodic Interrupt Timers

Communication Interfaces

- 3 x Low-Power I^2C
- Flex™ I/O
- 3 x LPUART
- Crystal-less USB FS Device/OTG
- 3 x Low-Power SPI
- USB Voltage Regulator
- ISO7816-E VMSIM
- Serial Audio Interface/I2S

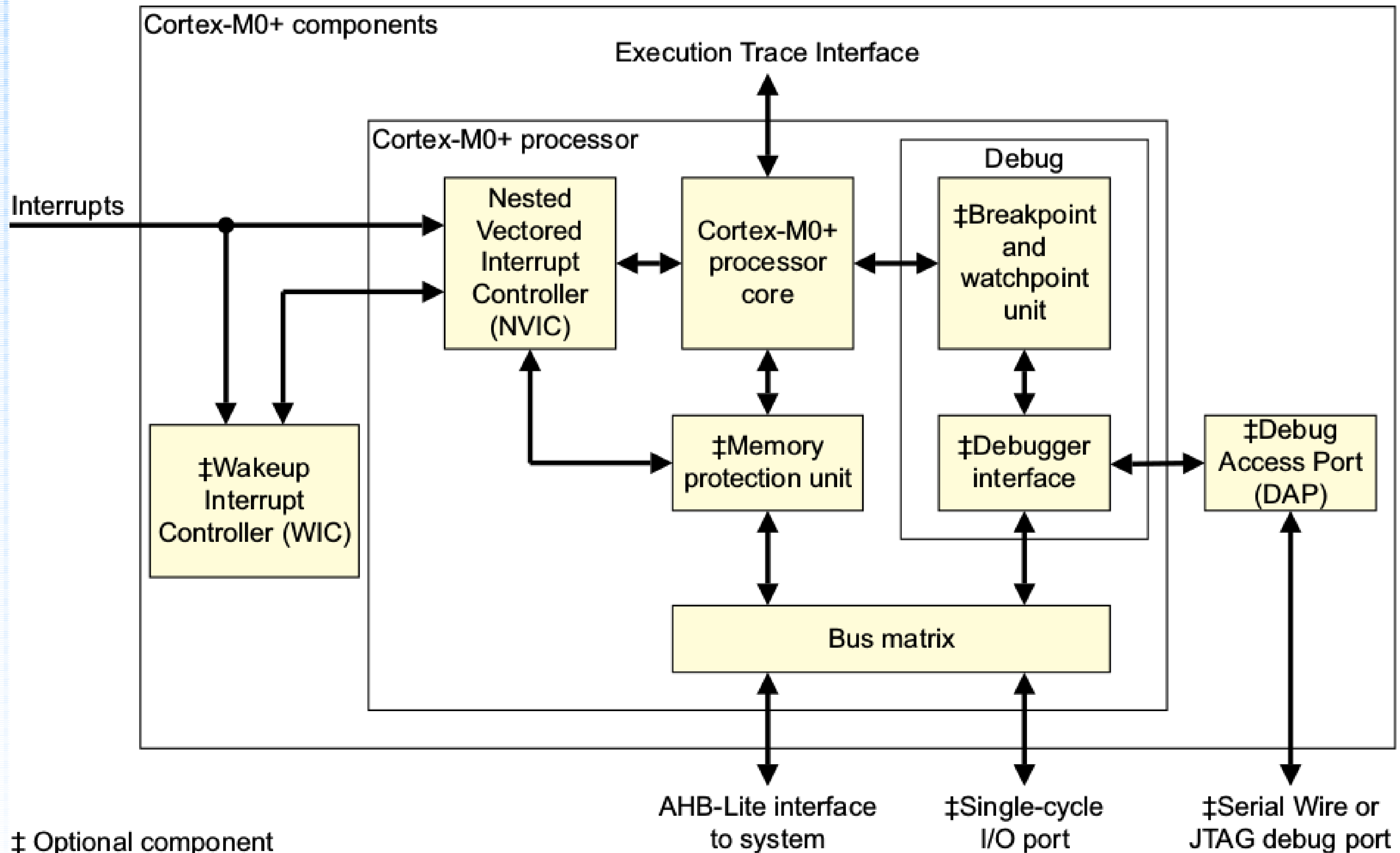
HMI

- GPIOs with Interrupts
- 16-ch. Touch Sensor Interface

Optional

Fonte: NXP

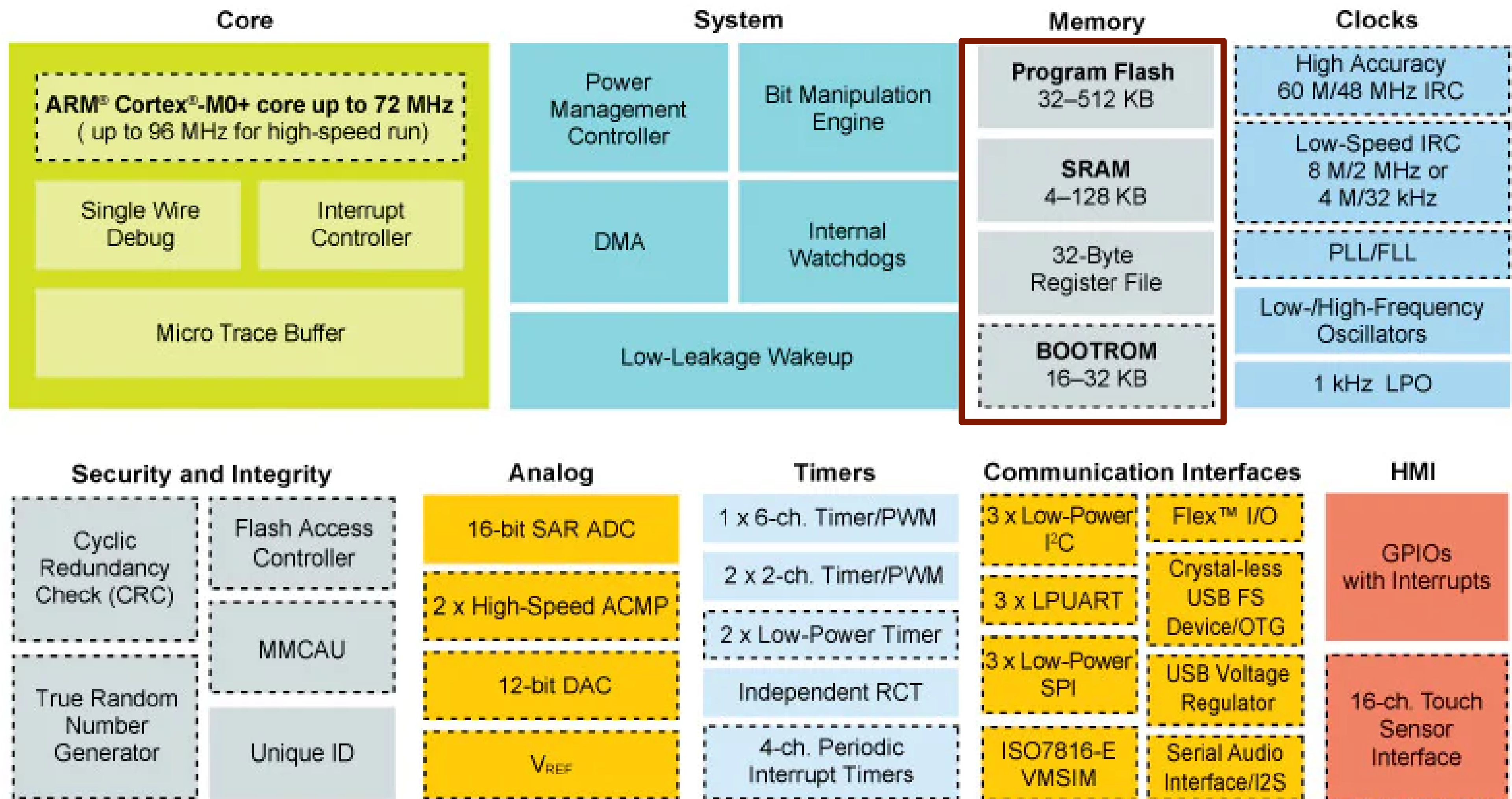
Processador ARM Cortex-M0+



ARMv6: Arquitetura ARM versão 6

- Arquitetura Von Neumann: dados e instruções compartilham o mesmo espaço de endereços.
- Arquitetura de E/S mapeada em memória.
- Arquitetura RISC: número reduzido de instruções de comprimento fixo (estado ARM: 32 *bits* e estado Thumb: 16 *bits*)
  - Subconjunto de instruções do repertório ARM (32 *bits*) recodificadas em 16 *bits* para aumentar a densidade de instruções.
- Arquitetura *Load*/*Store* ou registrador-registrador
  - Instruções de 2 endereços
- 13 registradores de propósito geral (R0-R12), 1 ponteiro da pilha (SP/R13), 1 registrador de retorno (LR/R14) 1 contador de programa (PC/R15), e registradores de estado do programa corrente (xPSR).
- Muitas instruções são executadas incondicionalmente, não necessariamente atualizam os quatro *bits* de condição.

Microcontrolador Kinetis L Séries KL2x



Fonte: NXP

Mapeamento das Unidades de Memória

Endereço	Região
0x0000_0000	FLASH (128K)
0x0002_0000	
0x0800_0000	
0x1FFF_F000	
0x2000_0000	SRAM (16K)
0x2000_3000	
0x4000_0000	
0x4008_0000	
0x400F_F000	
0x4010_0000	
0xF000_2000	ROM (4K)
0xF000_3000	
0xFFFF_FFFF	

Formato de Códigos Thumb

Table A5-1 shows the allocation of 16-bit instruction encodings.

Table A5-1 16-bit Thumb instruction encoding

opcode	Instruction or instruction class	
00xxxx	*Shift (immediate), add, subtract, move, and compare* on page A5-85	
010000	*Data processing* on page A5-86	
010001	*Special data instructions and branch and exchange* on page A5-87	
01001x	Load from Literal Pool, see *LDR (literal)* on page A6-141	Memória ↔ Processador
0101xx 011xxx 100xxx	*Load/store single data item* on page A5-88	Memória ↔ Processador
10100x	Generate PC-relative address, see *ADR* on page A6-115	
10101x	Generate SP-relative address, see *ADD (SP plus immediate)* on page A6-111	
1011xx	*Miscellaneous 16-bit instructions* on page A5-89	
11000x	Store multiple registers, see *STM, STMIA, STMEA* on page A6-175	Memória ↔ Processador
11001x	Load multiple registers, see *LDM, LDMIA, LDMFD* on page A6-137	Memória ↔ Processador
1101xx	*Conditional branch, and Supervisor Call* on page A5-90	
11100x	Unconditional Branch, see *B* on page A6-119	

Modos de Endereçamento

- Imediato

  movs %r1,#128 @r1 ← 128

- por registradores

  orr %r2,%r1 @r2 ← [r2]||[r1]

  lsl %r1,%r1,#1 @r1 ← [r1]<<1

- Indireto por registrador (base)

  add %sp,%sp,#8 @sp ← [sp]+8

- Relativo a PC

  ldr r0, [pc, #48] @r0 ←[pc+48]

CodeWarrior IDE Development Suite

Informações Adicionais

- *Instruction Set Architecture*:

  http://www.cs.umd.edu/~meesh/cmsc411/CourseResources/CA-online/chapter/instruction-set-architecture/index.html

- *RISC, CISC, and ISA Variations*:

  http://www.cs.cornell.edu/courses/cs3410/2018fa/schedule/slides/10-isa.pdf

- Documentações sobre KL2x da NXP

  https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/kl-series-cortex-m0-plus/kinetis-kl2x-72-96-mhz-usb-ultra-low-power-microcontrollers-mcus-based-on-arm-cortex-m0-plus-core:KL2x

Informações Adicionais

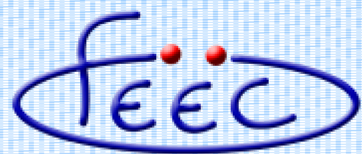
- *GNU ARM Assembler Quick Reference*

  ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/complementos/gnu-arm-directives.pdf
- *Introduction to ARM thumb*

  https://www.embedded.com/introduction-to-arm-thumb/
- *Thumb 16-bit Instruction Set Quick Reference Card*

  ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/ARM_QRC0006_UAL16.pdf
- *Alphabetic list of ARMv6-M Thumb instructions*

  Seção A6 em ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/ARMv6-M.pdf



EA871

Linguagem *Assembly*

Assembly embutido em C

Wu Shin – Ting
DCA – FEEC - Unicamp
Segundo Semestre de 2020

Assembly embutido em C

- Permite acessar diretamente às funções implementadas em *hardware*, que ainda não são disponíveis em linguagem de alto nível.
- É extremamente útil para implementação de *drivers* dos dispositivos e uso de *hardwares* altamente especializados.
- Porém, por demandar um bom domínio sobre o *hardware* envolvido, procure esgotar primeiro outras alternativas.

Sintaxe

```
asm asm-qualificadores (
códigos em assembly
: Operandos de saída
[:Operandos de entrada
[:Clobbers]])
```

```
asm asm-qualificadores (
códigos em assembly
:
: Operandos de entrada
: Clobbers
: GotoRotulos))
```

asm-qualificadores

- volatile
  - Desabilita algumas otimizações
- inline
  - Otimiza o tamanho dos códigos
- goto
  - Informa ao compilador da possível ocorrência de desvios

Códigos em *Assembly*

- *Strings* de instruções em *assembly*.
- Instruções devem ser separadas por caracteres de controle, como quebra de linha (‘\n’) e tab (‘\t’).
- Incluem tokens (acompanhados de %) que referem as entradas, saídas e os rótulos de desvio. Estes são substituídos pelo compilador.
- GCC não processa as instruções em assembly e não sabe os símbolos que eles referenciam, a menos que eles sejam listados explicitamente.

Operandos de Saída

- Lista de operandos separados por vírgulas.
- Formato de definição de cada variável

  [[nome_simbólico]] “restrições” (nome_da_variável)

  – nome simbólico: referenciado no código por

    %[nome_simbólico]. Quando não definido explicitamente, é usado alternativamente a referência posicional “%0”, “%1”, etc.

  – Restrições: <tipo de acesso><local de armazenamento>

    - tipo de acesso: ‘=’ (sobreescreve); ‘+’ (leitura e escrita)
    - local onde o operando reside: ‘r’ (registrador), ‘m’ (memória)
- Quando não já operando de saída, deve-se deixar a lista vazia.

Operandos de Entrada

- Lista de operandos separados por vírgulas.
- Formato de definição de cada variável

  [[nome_simbólico]] “restrições” (nome_da_variável)

  - nome simbólico: referenciado no código por

    %[nome_simbólico]. Quando não definido explicitamente, é usado alternativamente a referência posicional “%0”, “%1”, etc.
  - Restrições: <local de armazenamento>
    - local onde o operando reside: ‘r’ (registrador), ‘m’ (memória), ‘g’ (qualquer local)
    - Para especificar que duas variáveis, uma de entrada e uma de saída, devam ocupar a mesma posição de memória, usa-se as mesmas referências posicionais (numéricas).

Clobbers

- Lista de registradores, memória e registrador de condições, cujos conteúdos são alterados pelas instruções em *assembly*
- Os recursos devem estar declaradas entre aspas.
- Não devem ser incluídos os recursos usados pelos operandos de entrada e de saída.

CodeWarrior IDE Development Suite

Informações Adicionais

- *How to Use Inline Assembly Language in C Code*

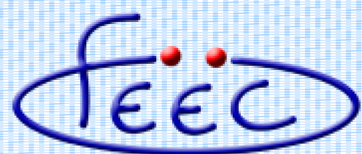
  https://gcc.gnu.org/onlinedocs/gcc/Using-Assembly-Language-with-C.html

- *ARM GCC Inline Assembler Cookbook*

  https://www.ic.unicamp.br/~celio/mc404-s2-2015/docs/ARM-GCC-Inline-Assembler-Cookbook.pdf

- *ARM Compiler User Guide Version 6.6*

  https://developer.arm.com/documentation/100748/0606/Using-Assembly-and-Intrinsics-in-C-or-C---Code/Writing-inline-assembly-code



EA871

Linguagem *Assembly*

Estimativa de tempo de execução

Analisador Lógico

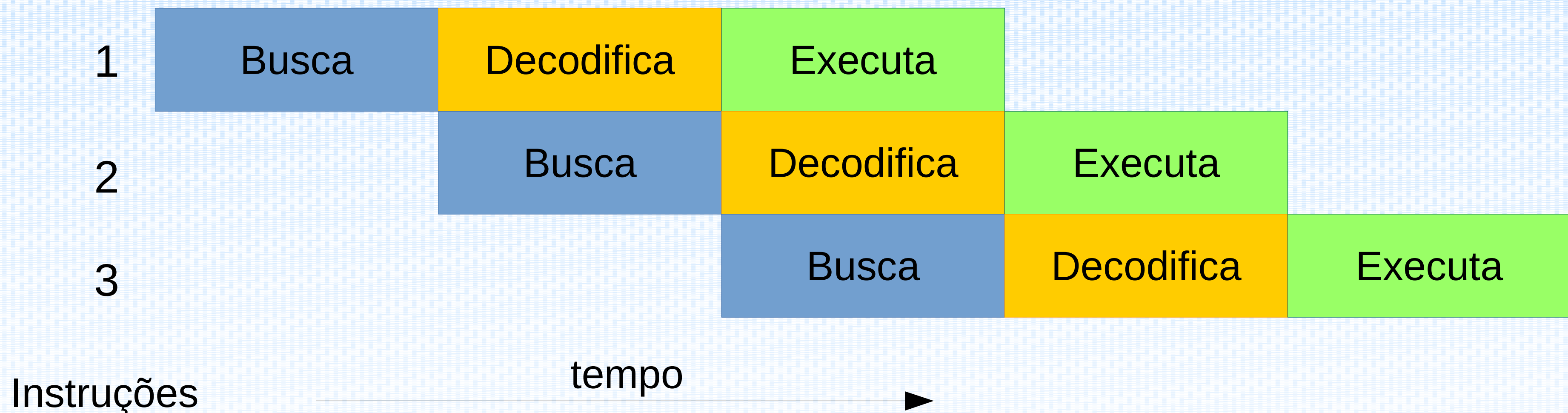
Wu Shin – Ting
DCA – FEEC - Unicamp
Segundo Semestre de 2020

Conceitos

- **Ciclo de instrução** ou ciclo busca-execução: consiste no intervalo de tempo, medido em ciclos de relógio, que o processador precisa para processar uma instrução endereçada pelo contador de programa (PC). Compreende 3 estágios:
  - **Busca** de instrução cujo endereço está no contador de programa (PC),
  - **Decodificação** de instrução em operandos e sinais de controle de operação, e
  - **Execução** de instrução e armazenamento do resultado.

Conceitos

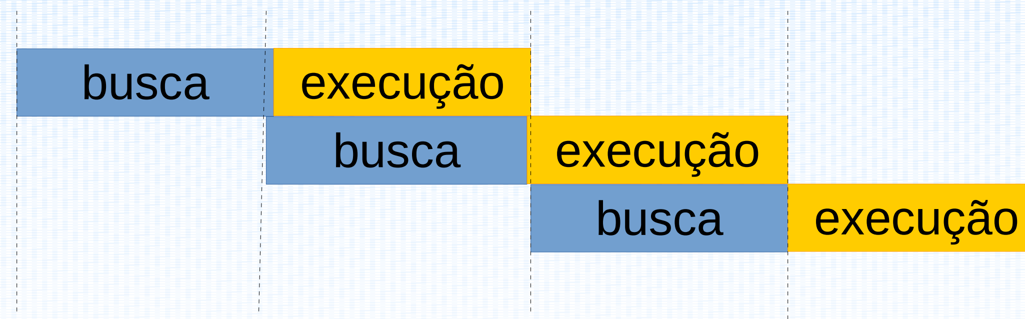
- **Segmentação de instruções** (*pipeline*): consiste essencialmente em dividir o processamento de uma instrução numa sequência de micro-instruções (estágios) e paralelizar a execução destas micro-instruções quando não há conflito no uso de recursos.



$$Ciclo\ de\ instrução\ ideal = \frac{Ciclo\ de\ instrução\ sem\ pipeline}{número\ de\ estágios}$$

ARM Cortex-M0+

- *Pipeline* de 2 estágios:
  - Busca de instrução cujo endereço está no contador de programa (PC),
  - Pré-decodificação: busca dos operandos nos registradores,
  - Pós-decodificação: cômputo de endereços efetivos na memória; ou
  - Cômputo da operação.



ARM Cortex-M0+

- Legenda para as tabelas de tempo em *clocks* das relógio.

a. Depends on multiplier implementation.
b. 2 if to AHB interface or SCS, 1 if to single-cycle I/O port
c. N is the number of elements in the list.
d. N is the number of elements in the list including PC or LR.
e. 2 if taken, 1 if not-taken.
f. Cycle count depends on processor and debug configuration.
g. Excludes time spent waiting for an interrupt or event.
h. Executes as NOP.

Fonte: Cortex-M0+ Technical Reference Manual

ARM Cortex-M0+

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
	Form address from SP	ADD Rd, SP, #<imm>	1
	Form address from PC	ADR Rd, <label>	1

Fonte: Cortex-M0+ Technical Reference Manual

ARM Cortex-M0+

Subtract	Lo and Lo	SUBS Rd, Rn, Rm	1
	3-bit immediate	SUBS Rd, Rn, #<imm>	1
	8-bit immediate	SUBS Rd, Rd, #<imm>	1
	With carry	SBCS Rd, Rd, Rm	1
	Immediate from SP	SUB SP, SP, #<imm>	1
Subtract	Negate	RSBS Rd, Rn, #0	1
Multiply	Multiply	MULS Rd, Rm, Rd	1 or 32[a]
Compare	Compare	CMP Rn, Rm	1
	Negative	CMN Rn, Rm	1
	Immediate	CMP Rn, #<imm>	1

Fonte: Cortex-M0+ Technical Reference Manual

ARM Cortex-M0+

Table 3-1 Cortex-M0+ instruction summary (continued)

Operation	Description	Assembler	Cycles
Logical	AND	ANDS Rd, Rd, Rm	1
	Exclusive OR	EORS Rd, Rd, Rm	1
	OR	ORRS Rd, Rd, Rm	1
	Bit clear	BICS Rd, Rd, Rm	1
	Move NOT	MVNS Rd, Rm	1
	AND test	TST Rn, Rm	1
Shift	Logical shift left by immediate	LSLS Rd, Rm, #<shift>	1
	Logical shift left by register	LSLS Rd, Rd, Rs	1
	Logical shift right by immediate	LSRS Rd, Rm, #<shift>	1
	Logical shift right by register	LSRS Rd, Rd, Rs	1
	Arithmetic shift right	ASRS Rd, Rm, #<shift>	1
	Arithmetic shift right by register	ASRS Rd, Rd, Rs	1

Fonte: Cortex-M0+ Technical Reference Manual

ARM Cortex-M0+

Rotate	Rotate right by register	RORS Rd, Rd, Rs	1
Load	Word, immediate offset	LDR Rd, [Rn, #<imm>]	2 or 1[b]
	Halfword, immediate offset	LDRH Rd, [Rn, #<imm>]	2 or 1[b]
	Byte, immediate offset	LDRB Rd, [Rn, #<imm>]	2 or 1[b]
	Word, register offset	LDR Rd, [Rn, Rm]	2 or 1[b]
	Halfword, register offset	LDRH Rd, [Rn, Rm]	2 or 1[b]
	Signed halfword, register offset	LDRSH Rd, [Rn, Rm]	2 or 1[b]
	Byte, register offset	LDRB Rd, [Rn, Rm]	2 or 1[b]
Load	Signed byte, register offset	LDRSB Rd, [Rn, Rm]	2 or 1[b]
	PC-relative	LDR Rd, <label>	2 or 1[b]
	SP-relative	LDR Rd, [SP, #<imm>]	2 or 1[b]
	Multiple, excluding base	LDM Rn!, {<loreglist>}	1+N[c]
	Multiple, including base	LDM Rn, {<loreglist>}	1+N[c]

Fonte: Cortex-M0+ Technical Reference Manual

ARM Cortex-M0+

Table 3-1 Cortex-M0+ instruction summary (continued)

Operation	Description	Assembler	Cycles
Store	Word, immediate offset	STR Rd, [Rn, #<imm>]	2 or 1[b]
	Halfword, immediate offset	STRH Rd, [Rn, #<imm>]	2 or 1[b]
	Byte, immediate offset	STRB Rd, [Rn, #<imm>]	2 or 1[b]
	Word, register offset	STR Rd, [Rn, Rm]	2 or 1[b]
	Halfword, register offset	STRH Rd, [Rn, Rm]	2 or 1[b]
	Byte, register offset	STRB Rd, [Rn, Rm]	2 or 1[b]
	SP-relative	STR Rd, [SP, #<imm>]	2 or 1[b]
	Multiple	STM Rn!, {<loreglist>}	1+N[c]
Push	Push	PUSH {<loreglist>}	1+N[c]
	Push with link register	PUSH {<loreglist>, LR}	1+N[d]
Pop	Pop	POP {<loreglist>}	1+N[c]
	Pop and return	POP {<loreglist>, PC}	3+N[d]

Fonte: Cortex-M0+ Technical Reference Manual

Branch	Conditional	B<cc> <label>	1 or 2[e]
	Unconditional	B <label>	2
	With link	BL <label>	3
	With exchange	BX Rm	2
	With link and exchange	BLX Rm	2
Extend	Signed halfword to word	SXTH Rd, Rm	1
	Signed byte to word	SXTB Rd, Rm	1
	Unsigned halfword	UXTH Rd, Rm	1
Extend	Unsigned byte	UXTB Rd, Rm	1
Reverse	Bytes in word	REV Rd, Rm	1
	Bytes in both halfwords	REV16 Rd, Rm	1
	Signed bottom half word	REVSH Rd, Rm	1
State change	Supervisor Call	SVC #<imm>	-[f]
	Disable interrupts	CPSID i	1
	Enable interrupts	CPSIE i	1
	Read special register	MRS Rd, <specreg>	3
	Write special register	MSR <specreg>, Rn	3
	Breakpoint	BKPT #<imm>	-[f]

Fonte: Cortex-M0+ Technical Reference Manual

ARM Cortex-M0+

Table 3-1 Cortex-M0+ instruction summary (continued)

Operation	Description	Assembler	Cycles
Hint	Send event	SEV	1
	Wait for event	WFE	2[g]
	Wait for interrupt	WFI	2[g]
	Yield	YIELD	1[h]
	No operation	NOP	1
Barriers	Instruction synchronization	ISB	3
	Data memory	DMB	3
	Data synchronization	DSB	3

Fonte: Cortex-M0+ Technical Reference Manual

Estimativa do tempo de execução

- Frequência do relógio do Cortex-M0+
  - f = 20,97152MHz

- Período
  - $T = \frac{1}{20{,}97152 \times 10^6} s$

- Tempo de execução de uma sequência de instruções
  - t = quantidade de ciclos * T

Fonte: Cortex-M0+ Technical Reference Manual

CodeWarrior IDE Development Suite

Informações Adicionais

- Cortex-M0+ Technical Reference Manual

  ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/Cortex-M0+.pdf

- Saleae Logic Analyzer on the micro:bit and the Arduino

  https://www.youtube.com/watch?v=tu7Af_ZCkeo