

Sistemas Encaixados

Tema 1: Introdución aos Sistemas Encaixados

Oscar García Lorenzo

Escola Politécnica Superior de Enxeñaría

Introdución aos Sistemas Encaixados

- 1 Introdución
- 2 Arquitectura Básica dun Sistema Encaixado
 - Programas
 - Arquitectura
 - Execución dun programa
- 3 Representación da Información
 - Representación
 - Aritmética
 - Cadeas de caracteres
 - Números Reais

Índice

1 Introducción

2 Arquitectura Básica dun Sistema Encaixado

- Programas
- Arquitectura
- Execución dun programa

3 Representación da Información

- Representación
- Aritmética
- Cadeas de caracteres
- Números Reais

Sistema encaixado

Que é?

Un sistema electrónico con, polo menos, un aparello controlador que está oculto ao usuario, é dicir, encaixado.

- Deseñados para cubrir necesidades específicas.
- Interactúan con periféricos:
 - Asincronicamente
 - Sincronicamente
- Poden ter que actuar en tempo real.
- Non soen ter o aspecto que se asocia a unha computadora.

Sistema encaixado

Que é?

Poden ser microondas, tostadoras, neveiras, lavadoras, televisores, routers inarámicos, semáforos, teléfonos móbiles, cámaras, reprodutores de Bluray, reloxos, aparellos como o rato, auriculares, teclados, etc.

- Antigamente (anos 1970) había que crear o controlador específicamente para cada aparello.
- Hoxe en día fabrícanse **microcontroladores** que pódense adaptar a diversas funcións.
- Incluso pódense usar microprocesadores más potentes para certas aplicacións.

Sistema encaixado

Microcontrolador

Un circuíto integrado programable, capaz de executar as ordes gravadas na súa memoria.

- Teñen os tres componentes principais dunha computadora:
 - Unidade central de procesamento (CPU)
 - Memoria
 - Comunicación con periféricos
- Deseñados para reducir o custo económico e uso de enerxía.
- Máis sinxelos que un microprocesador.
- Soen presentarse como SoC (System On a Chip), todos os componentes, incluíndo a memoria na mesma placa.
- Poden ter que programarse en ensamblador ou C (hoxe en día hai máis opcións).

Sistema encaixado

Microcontrolador

- ARM Cortex-M (Raspberry Pi Pico, Arduino Due)

Microprocesador

- ARM Cortex-A (Raspberry Pi 4)

Índice

1 Introducción

2 Arquitectura Básica dun Sistema Encaixado

- Programas
- Arquitectura
- Execución dun programa

3 Representación da Información

- Representación
- Aritmética
- Cadeas de caracteres
- Números Reais

Execución de programas

Como programamos?

Linguaxes de alto nivel: C, C++, Python ...

Que se executa?

Código máquina: código binario, ficheiros executables.

Execución de programas

Compilación

- O compilador traduce de linguaxe de alto nivel a código máquina.
 - Eficiente.
 - C, C++.

Interpretación

- O compilador traduce de linguaxe de alto nivel a bytecode, na execución.
 - O bytecode o executa unha máquina virtual pasándoo a código máquina.
 - Independente da plataforma.
 - Máis memoria e recursos.
 - Java, Python.



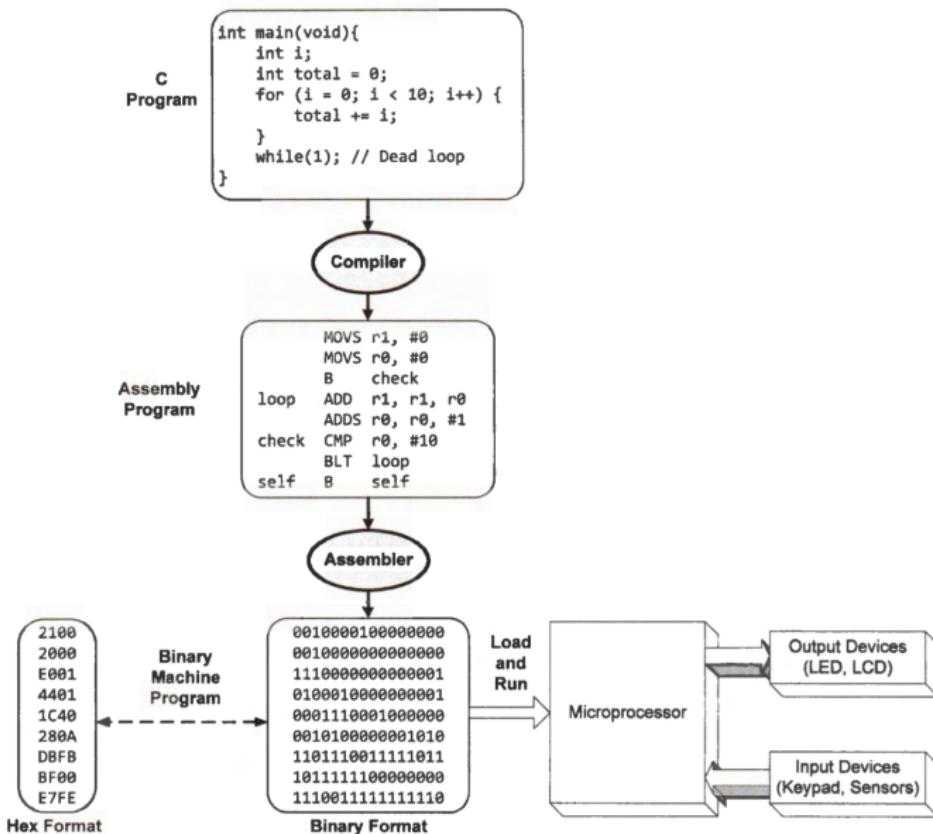
Execución de programas

Código ensamblador

Código de baixo nível, moi próximo ao código máquina.

- *Etiqueta*: representa unha dirección de memoria ou instrución do código.
 - *Mnemotécnico instrución*: operación a realizar, flexible para humanos.
 - *Operandos*: numéricos ou rexistros.
 - *Comentarios*
 - *Directivas ensamblador*: Non son instrucións, definen datos ou información para axudar ao ensamblador

Compilación



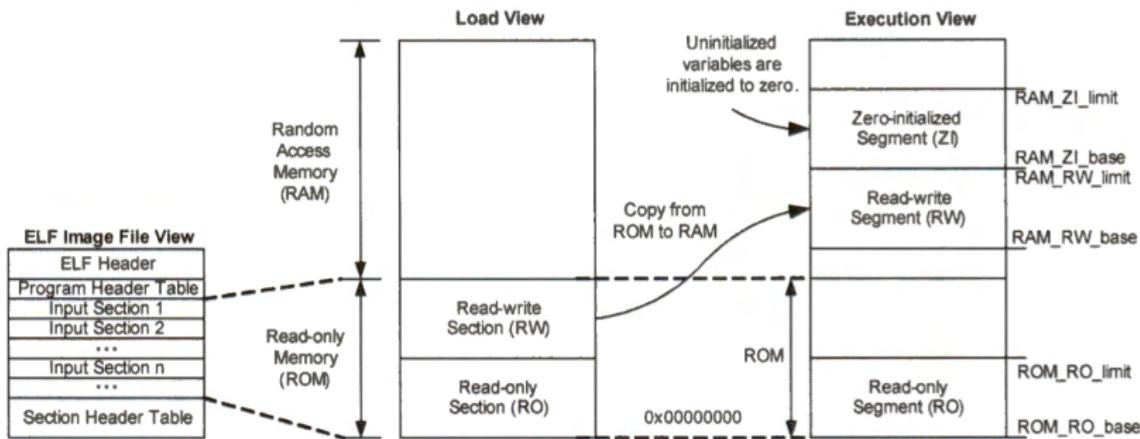
Executable

Ficheiro ELF

Hai diferentes tipos de ficherios, o ELF é un popular en UNIX.

- *Executable and Linkable Format.*
- *Interface enlazable:* usada para enlazar diferentes ficheiros ao compilar e executar.
- *Interface executable:* usada en tempo de execución para crear a imaxe a executar.
- *Vista de carga:* Define as direccións de memoria dos datos, de lectura e de lectura/escritura.
- *Vista de ejecución:* Informa ao procesador como cargar o programa na memoria.

ELF



Executar

- Os códigos soen gardarse en memoria non volátil (discos duros, memoria flash, tarxetas SD ...)
 - non se borra ao apagar o equipo.
- O procesador debe de cargar o programa á memoria principal (DRAM, SRAM ...)
 - borrase ao apagar o equipo.

Arquitecturas

Von Neumann

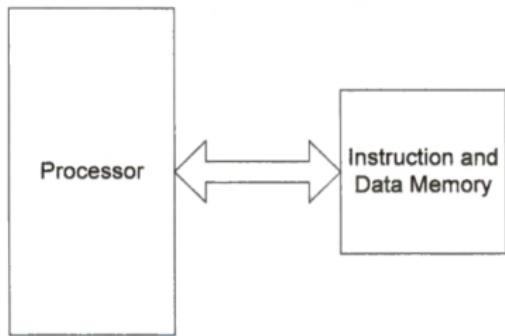
- Unha soa memoria pra instrución e datos.
- Relativamente barata e simple.
- A máis común hoxe en día en procesadores de uso xeral.*
 - *Aínda que dada a complexidade dos sistemas actuais, en realidade son unha combinación de Harvard e Von Neuman

Harvard

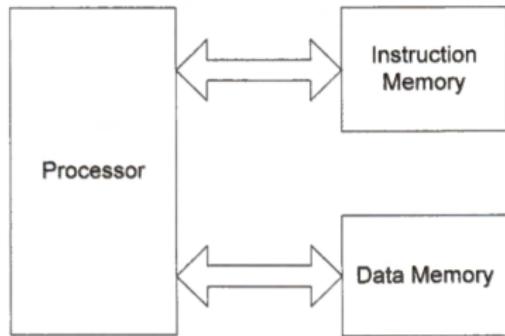
- Unha memoria pra instrucións e outra para datos.
- Pode ler á vez instrución e datos, más rápida.
- Máis eficiente enerxéticamente.
- Usase moito en microcontroladores.
- ARM Cortex-M3/M4/M7.

Arquitecturas

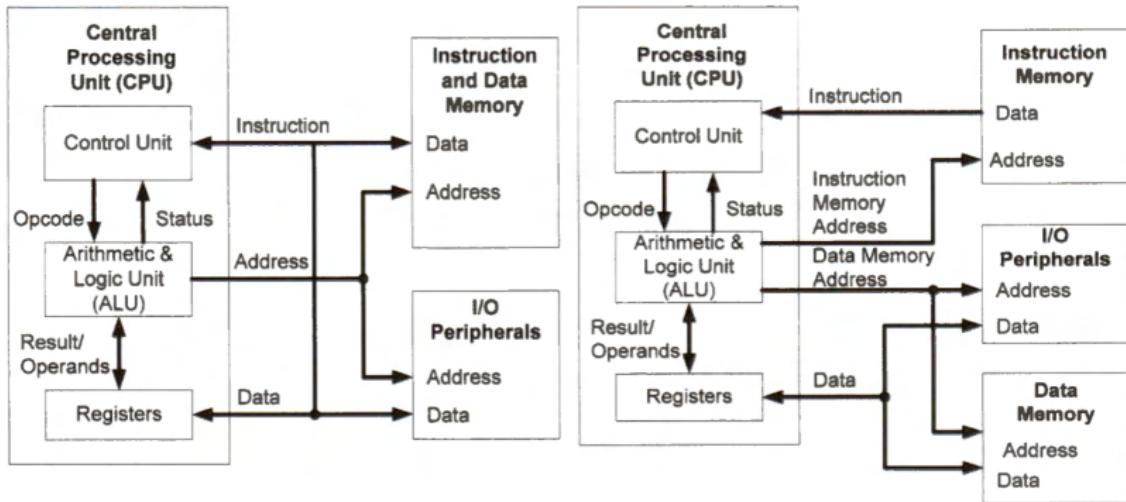
Von Neumann Architecture



Harvard Architecture



Arquitecturas



Memorias

Memoria direccionable

- As memorias direccionanse a nivel de Byte (8 bits).
- Os sistemas encaixados non soen pasar de 4 GB (e normalmente menos).

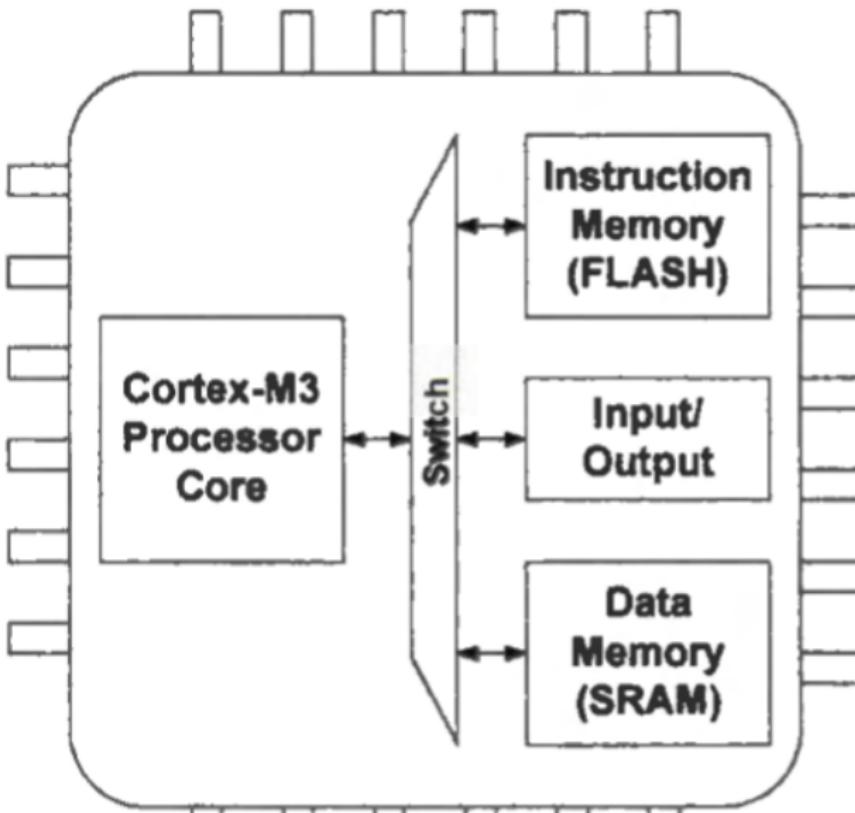
Name	Abbr.	Size
Kilo	K	$2^{10} = 1,024$
Mega	M	$2^{20} = 1,048,576$
Giga	G	$2^{30} = 1,073,741,824$
Tera	T	$2^{40} = 1,099,511,627,776$

Carga dun programa

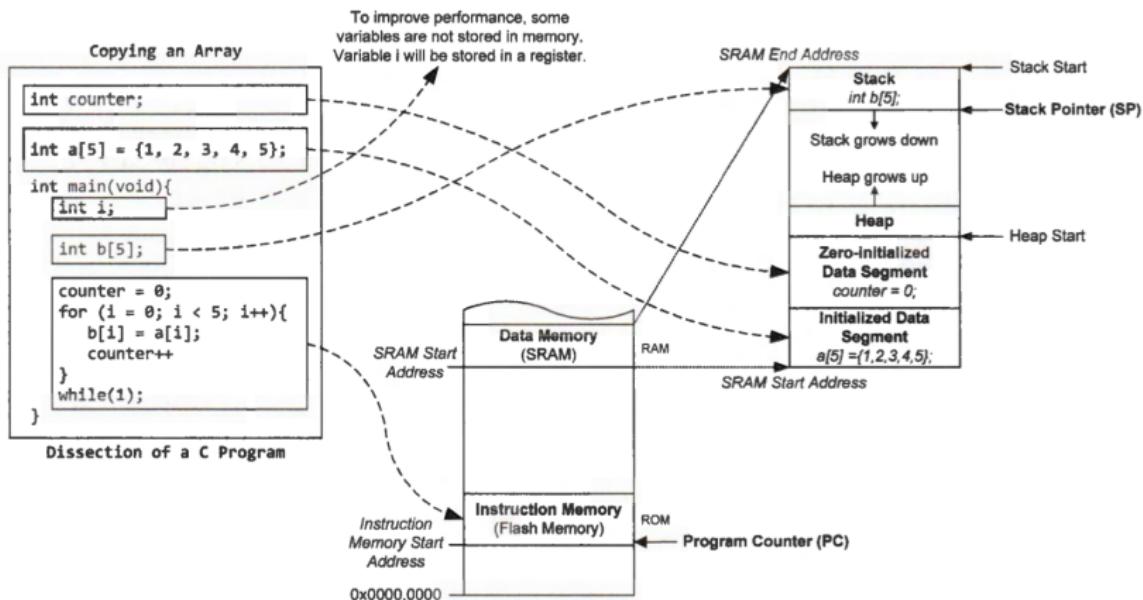
Memoria de datos

- *Datos inicializados*: variables estáticas e globais con valor.
- *Datos non inicializados*: variables estáticas e globais con 0.
- *Heap* (Montón): os datos creados durante a aplicación (malloc()).
- *Stack* (Pila): variables das subrutinas, serve para pasar datos as subrutinas.
 - O Heap e o Stack crecen en direccións opostas.

ARM Cortex-M3



ARM Cortex-M3

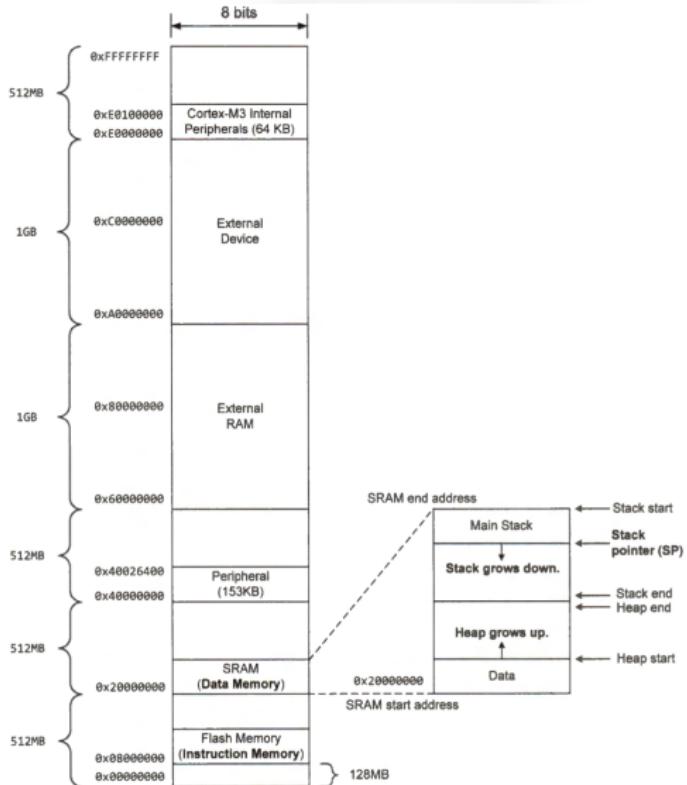


Carga dun programa

Memoria de periféricos

- Os periféricos, externos e internos, mapeanse á memoria.
- Así se pode acceder a eles comodamente.
- Poden ter:
 - Direccións de intercambio de datos.
 - Direccións de control.
 - Direccións de estado.
 - Memoria do dispositivo.

ARM Cortex-M3



Rexistros

Rexistros

- Serven para gardar valores.
- Poden ser de 16, 32, 64... bits.
- O procesador le ou escribe todos os bits á vez.
- Propósito xeral:
 - Gardan os operandos e resultados das operacións.
- Propósito especial:
 - Gardan información esencial para o funcionamento do procesador.
 - Contador de programa, indica en que instrución se está.

Rexistros

Reuso de Rexistros

- Acceder aos rexistros moito máis rápido que á memoria.
- Localidade temporal:
 - Sóense acceder sempre aos mesmos datos.
 - Se un dato se acaba de usar é probable que se volva a usar pronto.
- Localidade espacial:
 - Os datos soen estar xuntos.
 - Se se accede a un dato é probable que se acceda ao seguinte en memoria.
- Hai poucos rexistros:
 - Son rápidos, quéntanse.
 - Menos rexistros, menos bits para indicalos, instrución más curtas.
- Reuso de rexistros.

Rexistros

Alocación Rexistros

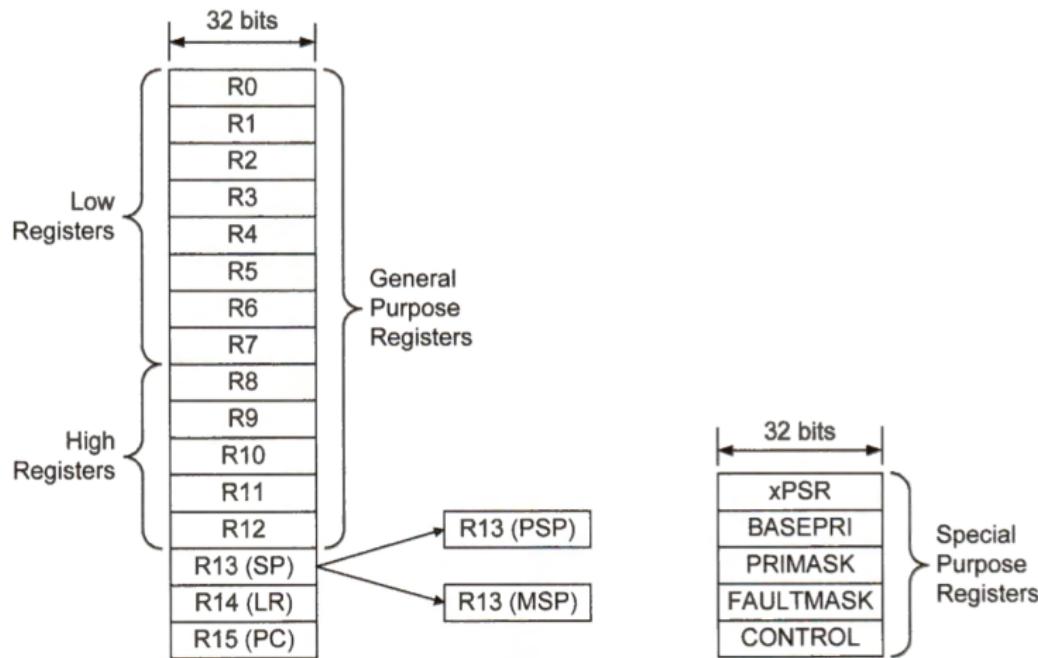
- Hai instrucións que so poden usar rexistros baixos.
- Hai instrucións que usan varios rexistros (multiplicación, 64 bit).
- Comprobar:
 - ver se as variables seguen vivas (se se volverán a usar).
 - se o tempo de vida de varias variables coincide, rexistros diferentes.
 - as variables que se acceden máis a miúdo aso rexistros, o resto á memoria.
- Os compiladores xa fan isto.

Rexistros

Distribución

- Rexistros baixos (r0-r7)
- Rexistros altos (r8-r12).
- Rregistro da pila (Stack Point, r13 (SP)):
 - Mantén a dirección do inicio da pila (stack).
 - En ARM Cortex hai 2 diferentes para programas xerais e interrupcións ou accesos privilexiados.
- Rregistro de enlace (Link Register, r14 (LR)).
 - Mantén a dirección da instrución posterior a unha chamada a subrutina, para volver ao programa principal.
- Contador de programa (Program Counter, r15 (PC)).
 - Dirección da próxima instrución a executar.
- Hai outros, xa os veremos cando xurdan.

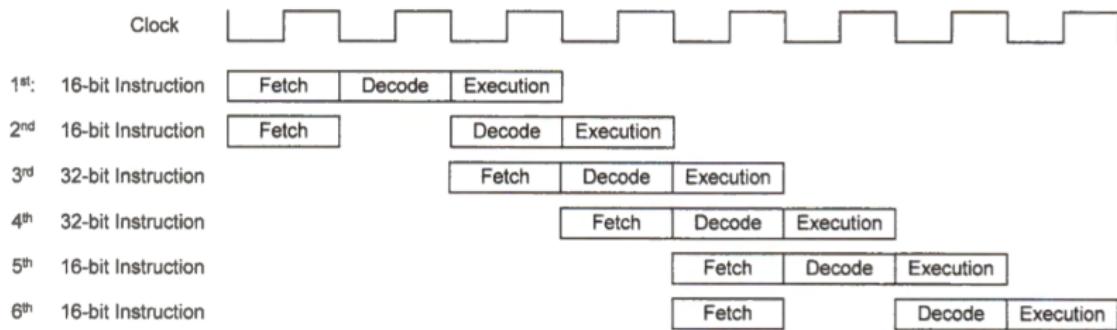
Rexistros



Execución

- As instrucións execútanse secuencialmente, se non se cambia o fluxo. (Non é así en procesadores más complexos.)
- O PC indica de onde se carga a seguinte instrución.
- O PC incrementase en 4 despois de cargar a instrución.
- Os Cortex teñen instrucións de 32 e 16 bit.
- Para os Cortex-M0/M3/M4 as instrución teñen 3 pasos:
 - Recuperar instrución. Colle 4 bytes da memoria de instrución e incrementa PC+4.
 - Descodificar instrución. Descobre que operación hai que realizar.
 - Executar instrución: Le os rexistros, opera con eles, accede a memoria, actualiza rexistros.
- Os pasos se poden realizar en pipeline para facer varias instrución ao mesmo tempo.

Rexistros



Pipelines

- Cortex M0+ só ten 2 pasos, recuperar e executar (Von Neumann).
- Cortex M7 ten múltiples pipelines, para load/store, operacións en punto flotante e dúas para operacións con enteros.
- Outros procesadores más complexos teñen múltiples pasos (e micro-instruccións) e pipelines, con hardware replicado para facer cousas ao mesmo tempo (por exemplo dúas operacións de suma á vez)

Programa Exemplo

C Program	Assembly Program	Machine Program	
		Binary	Hex
int a = 1;	AREA myData, DATA ALIGN a DCD 1	0000000000000000 0000000000000001	0x0000 0x0001
int b = 2;	b DCD 2	0000000000000000 0000000000000010	0x0000 0x0002
int c = 0;	c DCD 0	0000000000000000 0000000000000000	0x0000 0x0000
int main(){ c = a + b; while(1); }	AREA myCode, CODE EXPORT __main ALIGN ENTRY		
	__main PROC		
	LDR r1, =a	010010010000011	0x4903
	LDR r2, [r1]	0110100000001010	0x680A
	LDR r3, =b	010010110000011	0x4B03
	LDR r4, [r3]	0110100000011100	0x681C
	ADDS r5, r2, r4	0001100100011001	0x1915
	LDR r6, =c	010011100000011	0x4E03
	STR r5, [r6]	0110000000111001	0x6035
	stop B stop	1110011111111110	0xE7FE
	ENDP	0000000000000000	0x0000
	END	0010000000000000	0x2000
		0000000000001000	0x0004
		0010000000000000	0x2000
		0000000000001000	0x0008
		0010000000000000	0x2000

Compilar

- Os compiladores xeran o código máquina.
- O código máquina é para unha arquitectura en concreto.
- Diferentes compiladores poden xerar diferente código.
- Diferentes opcións (optimización) poden xerar diferente código.
- No caso anterior as instrucións ADDS é de 16 bits, por iso o compilador a usa en vez da ADD, aínda que non usa o flag S (xa o veremos).
- No código ensamblador pode haber pseudo-instruccións, que non corresponden exactamente ao código máquina, pero son más entendibles.

Memoria do programa

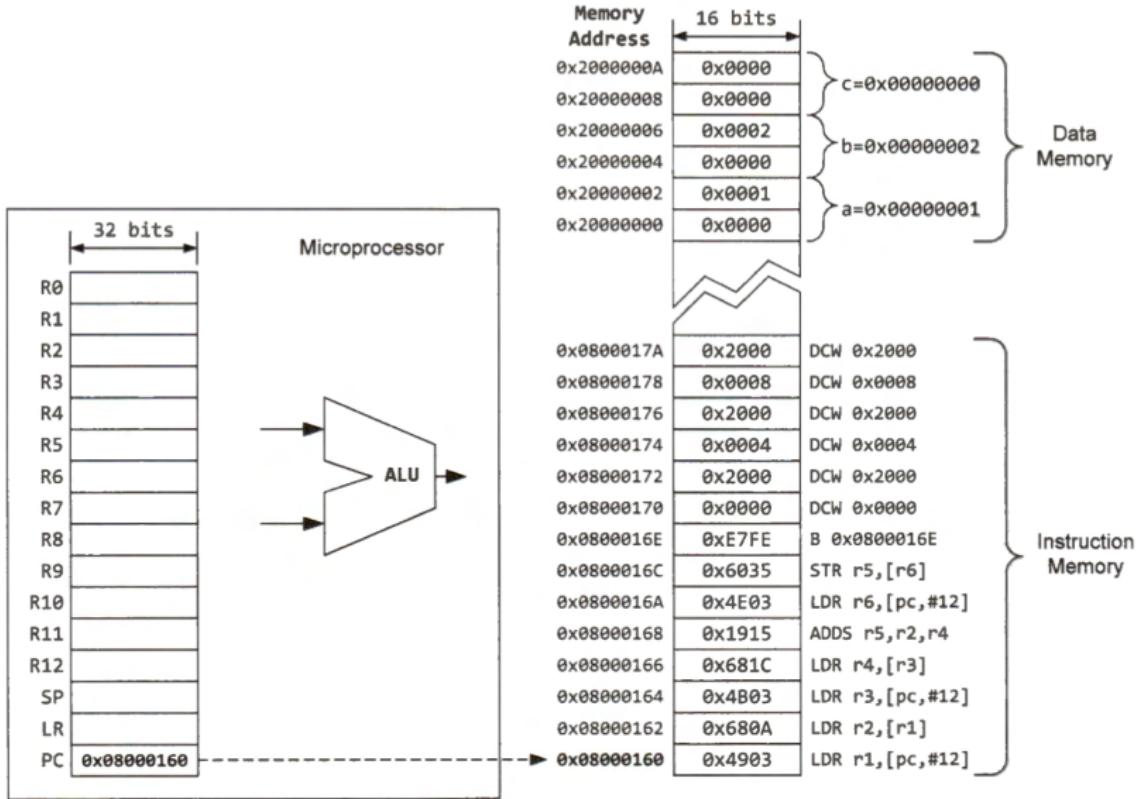
Memory Region	Memory Address	Binary Instruction	Assembly Instruction	Comments
Data Memory	0x20000000	0x0001	DCW 0x0001	
	0x20000002	0x0000	DCW 0x0000	; 0x00000001
	0x20000004	0x0002	DCW 0x0002	
	0x20000006	0x0000	DCW 0x0000	; 0x00000002
	0x20000008	0x0000	DCW 0x0000	
	0x2000000A	0x0000	DCW 0x0000	; 0x00000000
	
Instruction Memory	0x08000160	0x4903	LDR r1, [pc,#12]	; @0x08000170
	0x08000162	0x680A	LDR r2, [r1]	; r2 = a
	0x08000164	0x4B03	LDR r3, [pc,#12]	; @0x08000174
	0x08000166	0x681C	LDR r4, [r3]	; r4 = b
	0x08000168	0x1915	ADDS r5, r2, r4	; r5 = a + b
	0x0800016A	0x4E03	LDR r6, [pc,#12]	; @0x08000178
	0x0800016C	0x6035	STR r5, [r6]	; save c
	0x0800016E	0xE7FE	B 0x0800016E	; stop
	0x08000170	0x0000	DCW 0x0000	
	0x08000172	0x2000	DCW 0x2000	; 0x20000000
	0x08000174	0x0004	DCW 0x0004	
	0x08000176	0x2000	DCW 0x2000	; 0x20000004
	0x08000178	0x0008	DCW 0x0008	
	0x0800017A	0x2000	DCW 0x2000	; 0x20000008
	



Execución

- Cada instrucción tarda 3 ciclos nun Cortex-M3,4 (2 nun M0+)
- Neste exemplo as instrución son de 16 bits.
- O procesador ten un programa especial, o "boot loader" que inicializa o PC.
 - Empeza en MAIN() ou _MAIN.
- Os enterios ocupan 4 bytes en memoria.
- A Unidade Aritmética e Lóxica (ALU, en inglés) so pode operar con rexistros.
- Os rexistros pódense ler á vez.
- As variables en memoria teñen que traerse aos rexistros.
- **load-modify-store**
 - cargar variable desde memoria.
 - modificar/operar coa variable.
 - gardar a variable en memoria.

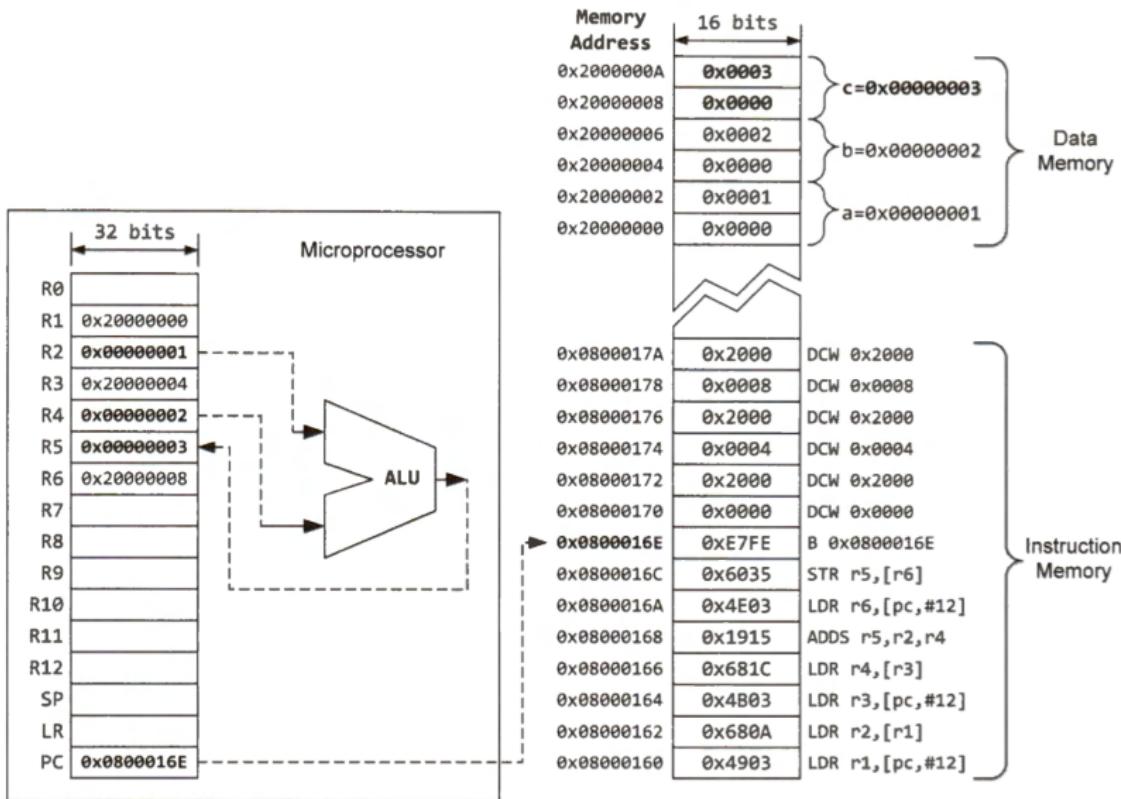
Memoria ao inicio



Execución

- Hai que ter coidado co PC, recordade que cada vez que se le incrementase en 4.
- As direccións das variables gárdanse na memoria de instrucións, para poder lelas.
- Ao final hai un bucle infinito, algo típico non sistemas encaixados.
- Non se pode operar directamente coa memoria.
- Non se pode gardar directamente na memoria.
- Todo isto quedará más claro nas prácticas.

Memoria ao inicio



Índice

1 Introducción

2 Arquitectura Básica dun Sistema Encaixado

- Programas
- Arquitectura
- Execución dun programa

3 Representación da Información

- Representación
- Aritmética
- Cadeas de caracteres
- Números Reais

Información en memoria

Bytes e palabras

E mellor acceder aos bits de varios en varios, na práctica:

- Bit = 1
 - Byte = 8 bits
 - Media-Palabra = Halfword = 2 bytes = 16 bits
 - Palabra = Word = 4 bytes = 32 bits
 - Doble-Palabra = Double-Word = 8 bytes = 64 bits
 - Poden usarse máis, pero non teñen nome.
-
- Bit más significativo (Most Significant Byte, MSB), más á esquerda
 - Bit menos significativo (Least Significant Byte, LSB), más á dereita

Información en memoria

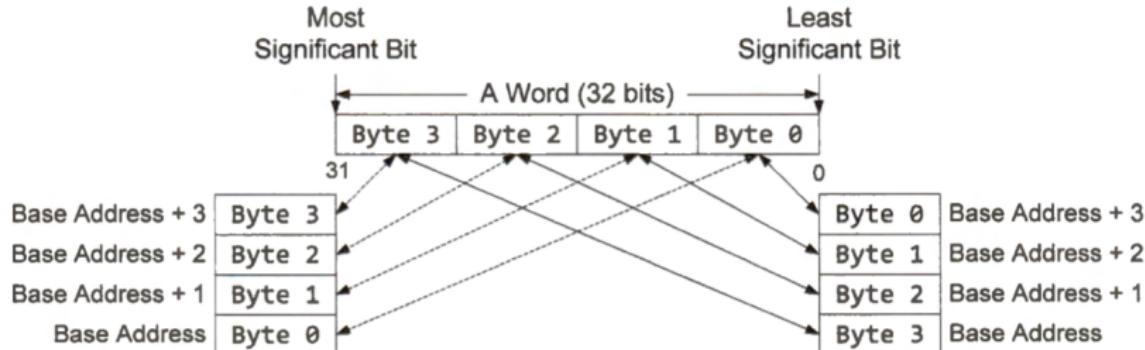
Á memoria accedese en bytes

Big Endian

O byte de maior orde gárdase na dirección más baixa (a parte grande ven primeiro)

Little Endian

O byte de menor orde gárdase na dirección más alta (a parte pequena ven primeiro)



Información en memoria e rexistros

- Os rexistros son de 32 bits.
- Para certos datos precisanse 2 rexistros.
- Para datos en memoria úsase a dirección más baixa para referirse a eles.

Basic data type of C language	Typical size in memory
char	1 byte
short	2 bytes or 1 halfword
signed/unsigned integer	4 bytes or 1 word
signed/unsigned long	4 bytes or 1 word
signed/unsigned long long	8 bytes or 1 double word
float	4 bytes or 1 word
double	8 bytes or 1 double word

Decimal-Binario-Hexadecimal

Decimal	Binary	Octal	Hexadecimal
0	0000	00	0x0
1	0001	01	0x1
2	0010	02	0x2
3	0011	03	0x3
4	0100	04	0x4
5	0101	05	0x5
6	0110	06	0x6
7	0111	07	0x7
8	1000	010	0x8
9	1001	011	0x9
10	1010	012	0xA
11	1011	013	0xB
12	1100	014	0xC
13	1101	015	0xD
14	1110	016	0xE
15	1111	017	0xF

Enteiros

Unsigned Integer

- Números representados: $[0, 2^n - 1]$

Para converter de decimal, dividir por 2 e ver os restos

$$\begin{array}{r}
 & & \text{Remainder} \\
 2 \overline{)52} & \cdots & 0 \\
 2 \overline{)26} & \cdots & 0 \\
 2 \overline{)13} & \cdots & 1 \\
 2 \overline{)6} & \cdots & 0 \\
 2 \overline{)3} & \cdots & 1 \\
 2 \overline{)1} & \cdots & 1 \\
 & & 0
 \end{array}$$

↑ LSB

MSB

Remainder

2	32	-----	0
2	16	-----	0
2	8	-----	0
2	4	-----	0
2	2	-----	0
2	1	-----	1
			0

LSB ↑

MSB ↓

Enteiros con signo

Signo e magnitud

MSB é o signo

- Números representados: $[-2^{n-1} + 1, 2^{n-1} - 1]$
- Dous ceros.

Complemento a 1

Un número negativo é inverter os bits do positivo

- Números representados: $[-2^{n-1} + 1, 2^{n-1} - 1]$
- Dous ceros.

Complemento a 2

Sumar 1 ao complemento 1

- Números representados: $[-2^{n-1}, 2^{n-1} - 1]$
- Un cero.

Enteiros con signo

Binary Bit String	Sign-and-Magnitude	One's Complement	Two's Complement
0000	+0	+0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	-0	-1

Enteiros con signo

Signo e magnitud

- Difícil de facer operacións en hardware.
- $00011(3) + 11011(-3) = 11110(-14)$
- Dúas representacións de cero.
- Non se usa.

Complemento a 1

- $\tilde{\alpha} = 2n - 1 - \alpha$.
- Fácil de obter, fácil operacións en hardware.
- $00001(+1) + 11110(-1) = 0000(0)$
- Usouse pero xa non.

Enteiros con signo

Complemento a 2

- $\bar{\alpha} = \tilde{\alpha} + 1 = 2n - \alpha$.
- $00110(+6) + 11010(-6) = 100000(0)$
- Un cero.
- Simplifica o hardware, non ten que preocuparse se son con ou sen signo.

Sumas e restas

Suma sen signo

- Empezase sumando o bit menos significativo.
- $0 + 0 = 0$, $0 + 1 = 1 + 0 = 1$
- $1 + 1 = 10 \leftarrow 1$ *acarreo*
- Se o último acarreo (*carry*) non é 0 o resultado non entra nos bits.

Resta sen signo

- Empezase restando o bit menos significativo.
- $0 - 0 = 1 - 1 = 0$, $1 - 0 = 1$
- $0 - 1 = 1 \leftarrow 1$ *prestamo*
- Se o último préstamo (*borrow*) non é 0 o resultado é negativo e non se pode representar.

Sumas e restas

Suma Complemento a 2

- Igual ao anterior.
- Pode ser más sinxelo que as restas sexan sumas do complemento a 2.
- O último acarreo desprezase se é igual ao penúltimo.
- Se o último acarreo e diferente ao penúltimo hai un desborde, o número non é representable.

Bandeiras de operacións

Bandeiras

Flags Son valores de 1 bit que indican algo sobre o resultado das operacións

- Rexistro APSR (application program status register)
- Acarreo (Carry) (C)
- Desborde (Overflow) (V)
- Cero (Zero) (Z)
- Negativo (Negative) (N)
- Saturación (Saturation) (Q)

Bandeiras de operacións

Acarreo/Carry (C)

Indica problemas con operacións con enteiros sen signo.

- É tamén o flag de borrow (B) para restas.
- Se é unha suma de enteiros sen signo $C=1$ se o número non é representable con 32 bits, senón $C=0$.
- Se é unha resta de enteiros sen signo $C=0$ se o número é negativo, senón $C=1$ (número positivo ou cero).
- $C = \text{NOT } B$

Bandeiras de operacións

Desborde/Overflow (V)

Indica problemas con operacións con enteros con signo.

- Suma:
 - Dous números positivos dan resultado negativo.
 - Dous números negativos dan resultado positivo.
- Resta:
 - Restar un número positivo a un negativo da resultado positivo.
 - Restar un número negativo a un positivo da resultado negativo.
- Máis fácil realizar sempre sumas e pasar a negativo o número que resta.
- Se o último acarreo é diferente ao penúltimo hai un desborde, o número non é representable.

Sumas e restas

- O complemento a 2 simplifica o hardware:
 - O mesmo sumador da resultado correcto:

Simple Addition (ignore the sign)	Unsigned Addition	Signed Addition	
$ \begin{array}{r} 1 \ 0 \ 1 \ 1 \ 1 \\ + \ 0 \ 0 \ 1 \ 1 \ 0 \\ \hline 1 \ 1 \ 1 \ 0 \ 1 \end{array} $	23	-9	
	+ 6	+ 6	

addend
 + addend
 sum

- O mesmo restador da resultado correcto:

Simple Subtraction (ignore the sign)	Unsigned Subtraction	Signed Subtraction	
$ \begin{array}{r} 1 \ 0 \ 1 \ 1 \ 1 \\ - \ 0 \ 0 \ 1 \ 1 \ 0 \\ \hline 1 \ 0 \ 0 \ 0 \ 1 \end{array} $	23	-9	
	- 6	- 6	

minuend
 - subtrahend
 difference

- O procesador non diferencia.
- As bandeiras C e V ponse para todas as operación, depende do programador saber que significan.
- En C (linguaxe) `unsigned int` e `int`, o compilador xa escolle ben.

Multiplicación de enteros

Multiplicación

- Mesmo hardware para con e sen signo.
 - Trunca os bits que sobran.
 - Non cambia os flags C e V.
 - Hai instrucións, que xa veremos, para multiplicacións grandes:
 - UMULL
 - SMULL

$$3 \times -3 = -9$$

$$3 \times 29 = 23?$$

División de enteiros

División

- Diferente hardware.
- Funciona con enteiros sen signo.
- Para enteiros con signo da problemas:
 - Converter a enteiros sen signo, dividir, volver a converter.
 - Operacións especiais, xa as veremos.
 - UDIV
 - SDIV

Resumo de operacións con enteros

- Sumas e restas, hardware non diferencia entre con e sen signo.
 - O hardware asume que son sen signo e actualiza C.
 - O hardware asume á vez que son con signo e actualiza V.
- Sumas e restas, programador interpreta.
 - Usar C se son sen signo.
 - Usar V se son con signo.
 - O compilador de C xa decide `unsigned int` e `int`
- Multiplicación, funciona se o resultado ten os mesmos bits que os operandos.
- División, hai que saber con que se opera.

Cadeas de caracteres

- Cada carácter ocupa 1 byte.
- ASCII, 7 bits
 - Antigo e obsoleto, pero sabes que vai funcionar en calquera sitio.
 - C e Ensamblador o usan.
- Nunha cadea se gardan os caracteres consecutivos en memoria, empezando pola dirección más baixa.
- Acaban en NUL.

ASCII

Dec	Hex	Char									
0	00	NUL	32	20	SP	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	\$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	,	71	47	G	103	67	g
8	08	BS	40	28	(72	48	H	104	68	h
9	09	HT	41	29)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	Ø	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	
29	1D	GS	61	3D	=	93	5D]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

ASCII

Memory Address	Memory Content	Letter
str + 12 →	0x00	\0
str + 11 →	0x79	y
str + 10 →	0x6C	l
str + 9 →	0x62	b
str + 8 →	0x6D	m
str + 7 →	0x65	e
str + 6 →	0x73	s
str + 5 →	0x73	s
str + 4 →	0x41	A
str + 3 →	0x20	space
str + 2 →	0x4D	M
str + 1 →	0x52	R
str →	0x41	A

“j” < “jar” < “jargon” < “jargonize”

“CAT” < “Cat” < “DOG” < “Dog” < “cat” < “dog”

“12” < “123” < “2” < “AB” < “Ab” < “ab” < “abc”

Punto Flotante

- Aritmética de punto fixo.
 - As cifras decimais fixas.
- Aritmética de punto flotante
 - As cifras decimais variables.
- Todas requiren hardware especial (Floating Point Unit, FPU).
- Moitos sistemas encaixados carecen del.
- Máis lenta e problemática.
- Evitala se se pode.

Referencias

Imaxes desde:

Y. Zhu. "Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C". Third Edition. E-Man Press LLC. 2017. ISBN-13: 978-0982692660