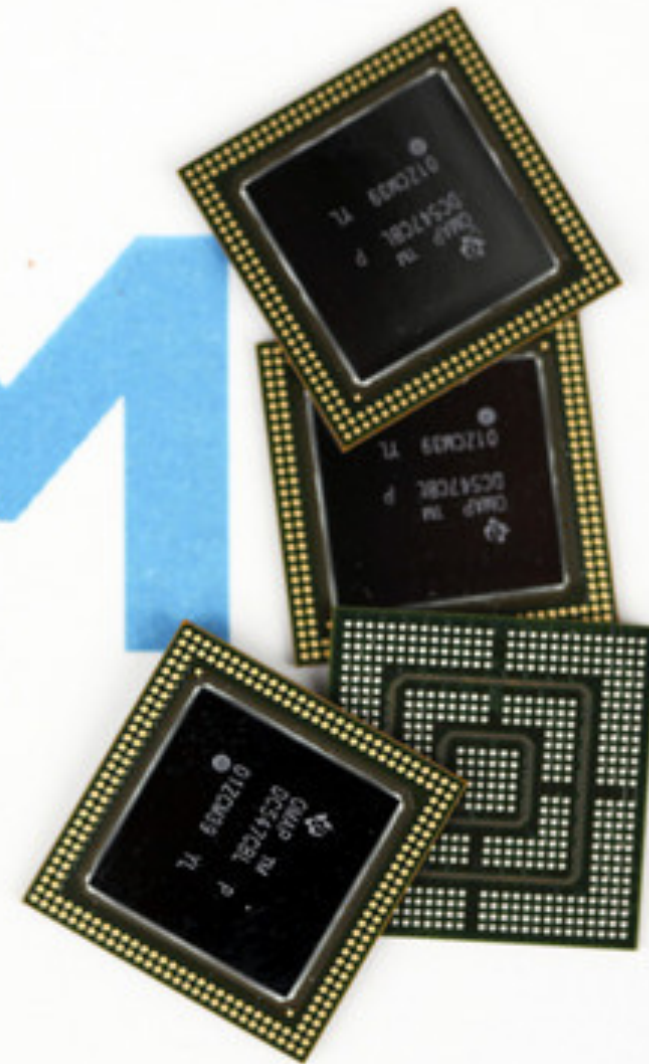


LINGUAGEM DE MÁQUINA

ARM EDITION

ARM



LINGUAGEM DE MÁQUINA

O QUE É?

- O **Assembly** é uma notação legível por humanos para o código de máquina que uma arquitetura de computador usa.

Machine code		Assembly Language	
:0:	6004000	:again:	OUT :x:
:1:	4004005		SUB :x: :y:
:2:	8010000		JMP GEZ :again:
:3:	0000000		HLT
:4:	10	:x:	(10)
:5:	1	:y:	(1)

LINGUAGEM DE MÁQUINA

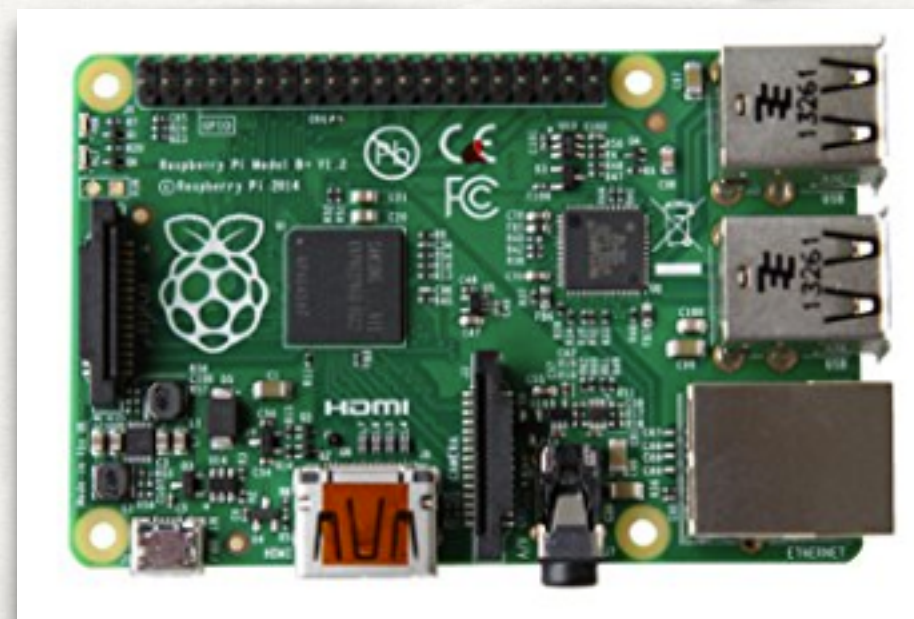
POR QUE APRENDER?

- **Assembly dá acesso direto ao hardware:**
 - Nem todo equipamento é perfeito para se trabalhar como um computador.
 - Existem dispositivos sem teclado, sem tela, com tipos diferentes de memória, de processador e com arquiteturas e funcionamentos diferentes.
- **Assembly serve para aprender como as coisas funcionam:**
 - *“Não sou louco em querer entender como funciona um carro. Louco é quem senta em cima de toneladas de ferro que possuem a força de centenas de cavalos, sem saber o que está acontecendo logo abaixo.”*
Anônimo

LINGUAGEM DE MÁQUINA

O QUE É ARM?

- ARM, originalmente Acorn RISC Machine, e depois Advanced RISC Machine, é uma família de arquiteturas RISC desenvolvida pela empresa britânica ARM Holdings.



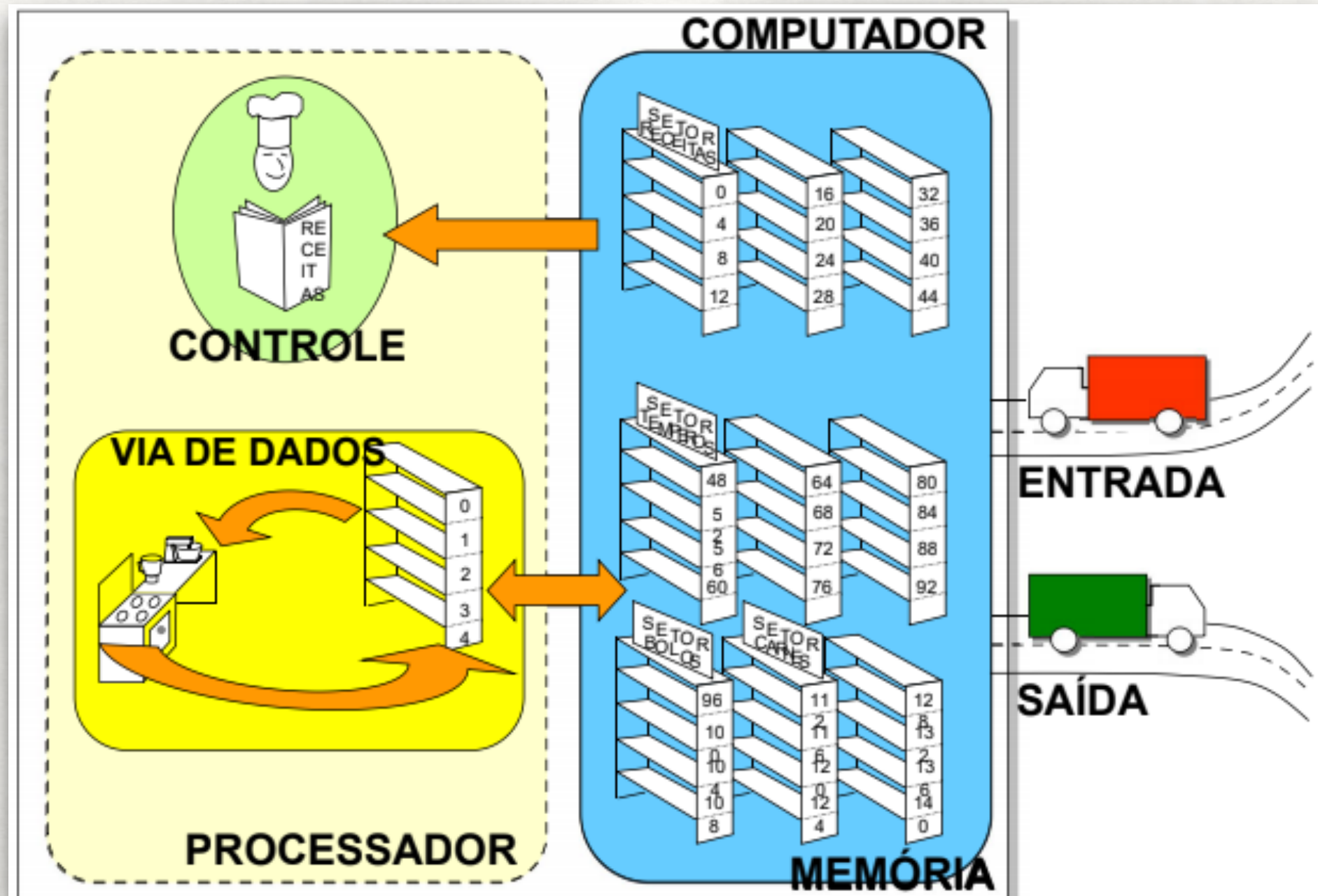
LINGUAGEM DE MÁQUINA

COMPUTADORES RISC

- **Reduced Instruction Set Computer** ou em português, **"Computador com um conjunto reduzido de instruções"**.
- Consiste em uma linha de arquitetura de processadores que favorece um conjunto simples e pequeno de instruções. Suas principais características são:
 - Banco de registrador com muitos registradores
 - Arquitetura Load/store
 - Endereçamento simples
 - Tamanho padrão nos campos de instrução

LINGUAGEM DE MÁQUINA

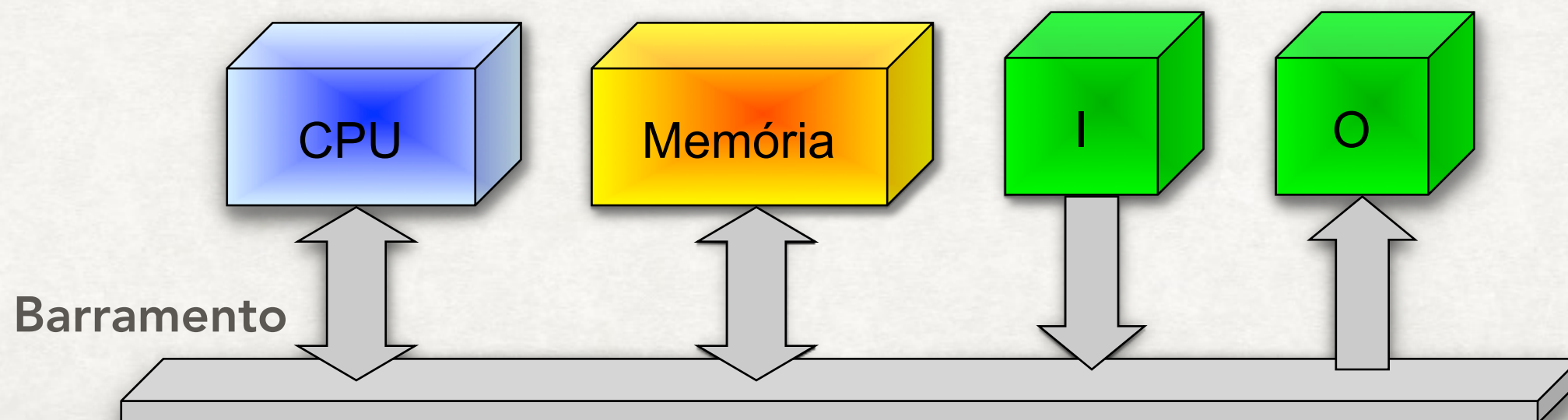
ORGANIZAÇÃO DE UM COMPUTADOR



LINGUAGEM DE MÁQUINA

ORGANIZAÇÃO DE UM COMPUTADOR

- Modelo de Von Neumann (1945)
 - Conceito de programa armazenado
 - Separação da Unidade Aritmética e de Controle
 - Utilização de barramentos e registradores
 - Hardware de entrada e saída (I/O)



LINGUAGEM DE MÁQUINA

COMO FUNCIONA NO ARM

- Temos em ARM que:
 - Byte (8 bits)
 - Meia palavra (*Halfword*) = 16 bits
 - Palavra (*Word*) = 32 bits
- O tamanho ocupado por qualquer instrução em ARM é de 32 bits.
- A interconexão dos componentes se dá por meio do barramento.

LINGUAGEM DE MÁQUINA

COMO FUNCIONA NO ARM

ARM Instruction Set Format

31	2827				1615				87				0				Instruction type																				
Cond	0	0	I	Opcode				S	Rn				Rd				Operand2				Data processing / PSR Transfer																
Cond	0	0	0	0	0	0	0	A	S	Rd				Rn				Rs				1	0	0	1	Rm				Multiply							
Cond	0	0	0	0	0	1	U	A	S	RdHi				RdLo				Rs				1	0	0	1	Rm				Long Multiply (v3M / v4 only)							
Cond	0	0	0	1	0	B	0	0	Rn				Rd				0				0	0	0	1	0	0	1	Rm				Swap					
Cond	0	1	I	P	U	B	W	L	Rn				Rd				Offset								Load/Store Byte/Word												
Cond	1	0	0	P	U	S	W	L	Rn				Register List												Load/Store Multiple												
Cond	0	0	0	P	U	1	W	L	Rn				Rd				Offset1				1	S	H	1	Offset2				Halfword transfer : Immediate offset (v4 only)								
Cond	0	0	0	P	U	0	W	L	Rn				Rd				0				0	0	0	1	S	H	1	Rm				Halfword transfer: Register offset (v4 only)					
Cond	1	0	1	I	Offset															Branch																	
Cond	0	0	0	1	0				0	1	0	1				1	1	1	1				1	1	1	0				0	0	1	Rn				Branch Exchange (v4T only)
Cond	1	1	0	P	U	N	W	L	Rn				CRd				CPNum				Offset								Coprocessor data transfer								
Cond	1	1	1	0	Op1				CRn				CRd				CPNum				Op2				0	CRm				Coprocessor data operation							
Cond	1	1	1	0	Op1				L	CRn				Rd				CPNum				Op2				1	CRm				Coprocessor register transfer						
Cond	1	1	1	1	SWI Number															Software interrupt																	

LINGUAGEM DE MÁQUINA

REGISTRADORES

- O ARM7 tem 37 registradores, sendo todos de 32 bits. Entre eles temos:
 - 1 dedicado para program counter (PC)
 - 1 dedicado para a posição de retorno de função (LR)
 - 1 dedicado para acompanhar o crescimento da pilha de execução (SP)
 - 5 dedicados para status de programas salvos
 - 30 de uso geral
 - 20 entre eles são registradores ocultos ao programa (conhecidos por Banked Out Registers)

LINGUAGEM DE MÁQUINA

REGISTRADORES

- Registradores de uso especial:
 - **SP:** Stack Pointer conhecido como ponteiro de pilha é o registrador que guarda a atual posição das variáveis e parâmetros durante as chamadas de função.
 - **LR:** Link Register é usado para guardar endereços de retorno para uma chamada de função.
 - **PC:** Program Counter é um registrador que é incrementado em 4 bytes toda vez que uma instrução é executada, mantendo a CPU informada do endereço da próxima instrução. Ele serve de base para cálculos de saltos e rotinas de um programa.

LINGUAGEM DE MÁQUINA

REGISTRADORES

Current Visible Registers

User Mode

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)
cpsr

Banked out Registers

Abort

FIQ

IRQ

SVC

Undef

	r8			
	r9			
	r10			
	r11			
	r12			
r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)
spsr	spsr	spsr	spsr	spsr

LINGUAGEM DE MÁQUINA

REGISTRADORES

- Na arquitetura ARM, existem seis principais modos de funcionamento:
 - **User:** Modo sem privilégios, maior parte dos programas usa apenas este modo
 - **System:** Modo com privilégios acessando os mesmos registradores que o modo Usuário
 - **FIQ:** Ativado quando uma interrupção de alta prioridade é ordenada
 - **IRQ:** Ativado quando uma interrupção de baixa prioridade é ordenada
 - **Supervisor:** Ativado durante um reset ou quando uma instrução de interrupção de programa é ativada
 - **Abort:** Ativado para lidar com violações de acesso de memória

LINGUAGEM DE MÁQUINA

MEMÓRIA

- Pode ser vista como um grande arranjo de células:
 - Onde informações são armazenadas (Store)
 - Onde informações são buscadas (Load)
- Tais operações são realizadas sobre uma palavra por vez.
- Esta arquitetura define palavras de 4 bytes de tamanho.

LINGUAGEM DE MÁQUINA

PROGRAMANDO

- É importante entender que:
 - Em assembly, estamos manipulando **registradores** do processador
 - Em código C (sem compilação), estamos manipulando **posições da memória**
 - A associação entre posições da memória e registradores é realizada pelo **compilador C**

LINGUAGEM DE MÁQUINA

PROGRAMANDO: DECLARANDO VARIÁVEIS

- Seu Bill é dono de uma padaria. Ele não tem muito dinheiro então possui apenas um Raspberry com 4 registradores. Ele decidiu reservar um dos registradores para acumular todo o dinheiro que recebe das suas vendas. Precisamos guardar em uma variável o valor que Bill recebeu no final do dia antes de podermos gravar no registrador. Como vamos fazer isso?



LINGUAGEM DE MÁQUINA

PROGRAMANDO: DECLARANDO VARIÁVEIS

- Usando a instrução **DCD** (**D**eclare **W**ord in **M**emory), é possível criar uma variável e já associar um valor a ela
- No exemplo vamos criar uma variável de nome **ganhoDoDia** e vamos associar a ela o valor de 74 reais.

ganhoDoDia

DCD

74

LINGUAGEM DE MÁQUINA

PROGRAMANDO: TRABALHANDO COM VARIÁVEIS

- Criamos agora uma variável na memória com nome e valor associado. Porém como vamos saber o endereço desse dado?
- A instrução **ADR (Address Load)** guarda em um registrador o endereço de memória de uma variável. Então, para guardar em R1 o endereço da variável *ganhoDoDia*, basta:

ADR R1 , *ganhoDoDia*

LINGUAGEM DE MÁQUINA

PROGRAMANDO: TRABALHANDO COM VARIÁVEIS

```
1 ganhoDoDia      DCD      74
2                 ADR      R1, ganhoDoDia
```

R1

256

Dec

Lembre-se: ADR guarda o endereço, não o valor da variável.

LINGUAGEM DE MÁQUINA

PROGRAMANDO: TRABALHANDO COM VARIÁVEIS

- Após obter o endereço da variável *ganhoDoDia*, podemos carregar o valor dela usando o comando **LDR (Load Register)**. Para isso precisamos informar:
 - O registrador que vai armazenar o valor da variável (**RD, Registrador de destino**)
 - O registrador que possui o endereço base da variável (**RN, Registrador de origem**)
 - O offset (indica a palavra desejada a partir do endereço base)

LDR RD, [RN, #OFFSET]

LINGUAGEM DE MÁQUINA

PROGRAMANDO: TRABALHANDO COM VARIÁVEIS

```
1 ganhoDoDia      DCD      74
2                  ADR      R1, ganhoDoDia
3                  LDR      R2,      [R1, #0]
```

R1	256	Dec
R2	74	Dec

LINGUAGEM DE MÁQUINA

PROGRAMANDO: TRABALHANDO COM VARIÁVEIS

- No Assembly ARM, cada palavra é composta por 4 bytes
- A instrução **DCD** pode ser usada para criar vetores
- Para demonstrar, vamos criar um vetor chamado **ganhoSemana**, que armazena os ganhos de segunda, terça, quarta e quinta da padaria de seu Bill

ganhoSemana DCD 12, 10, 56, 76

LINGUAGEM DE MÁQUINA

PROGRAMANDO: TRABALHANDO COM VARIÁVEIS

- Se quisermos carregar a segunda palavra do nosso vetor, vamos selecionar ela utilizando o **offset**
- Cada palavra possui 4 bytes e o endereço da primeira está na posição **#0**. Assim, para selecionar a segunda palavra, utilizamos:

LDR RD , [RN , #4]

```
1 ganhoDoDia      DCD      74
2 ganhoSemana     DCD     12,10,56,76
3                 ADR      R9, ganhoSemana
4                 ADR      R1, ganhoDoDia
5                 LDR      R2, [R9,#4]
```

R0	0	Dec	Bin	Hex
R1	256	Dec	Bin	Hex
R2	10	Dec	Bin	Hex

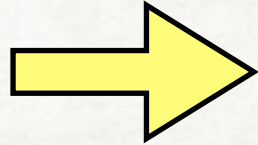
LINGUAGEM DE MÁQUINA

INSTRUÇÕES ARITMÉTICAS

Tipo de operação	Instrução	Exemplo
Soma	<code>ADD RD, RN, RM</code> <code>ADD RD, RD, #Int</code>	$RD = RN + RM$ $RD = RD + Int$
Subtração	<code>SUB RD, RN, RM</code> <code>SUB RD, RD, #Int</code>	$RD = RN - RM$ $RD = RD - Int$
Subtração inversa	<code>RSB RD, RN, RM</code> <code>RSB RD, RD, #Int</code>	$RD = (-RN) + RM$ $RD = (-RD) + Int$

LINGUAGEM DE MÁQUINA

INSTRUÇÕES LOAD/STORE

Load: Memória  Registrador (CPU)

Store: Registrador  Memória

Load Register: `LDR RD, [RN]`

Store Register: `STR RD, [RN]`

value at `[address]` found in `Rb`
is loaded into register `Ra`

`LDR` `Ra`, `[Rb]`
`STR` `Ra`, `[Rb]`

value found in register `Ra`
is stored to `[address]` found in `Rb`

LINGUAGEM DE MÁQUINA

INSTRUÇÕES LÓGICAS

- Existem instruções lógicas em Assembly ARM semelhantes às implementadas nas linguagens de alto nível. Alguns exemplos são:
 - E Bit a Bit: `AND RD, RN, RM`
 - Ou Bit a Bit: `ORR RD, RN, #Int`
 - OU Exclusivo: `EOR RD, RN, #Int`

LINGUAGEM DE MÁQUINA

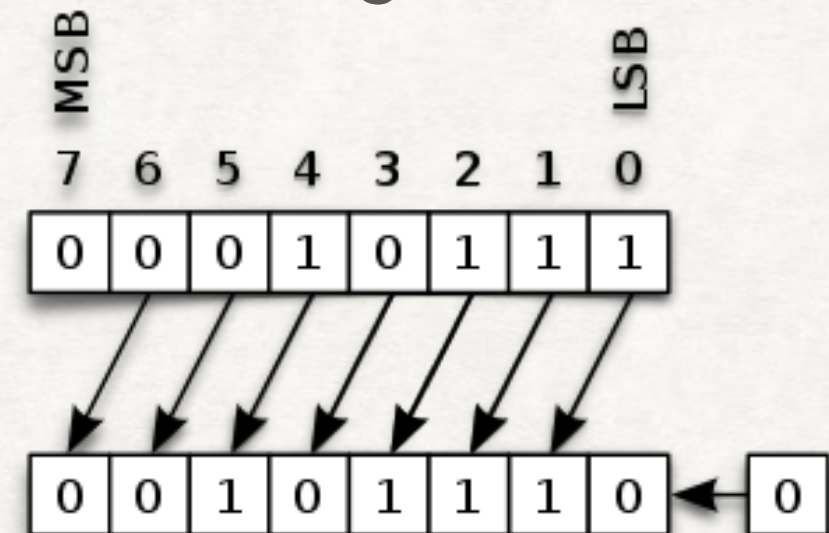
INSTRUÇÕES LÓGICAS

- O Assembly ainda oferece ao programador alterar os bits de um registrador da seguinte maneira:
 - Shift Lógico à esquerda: `LSL RD, RN, #Int`
 - Shift Lógico à direita: `LSR RD, RN, #Int`
 - Shift Aritmético à direita: `ASR RD, RN, #Int`
 - Rotação à direita: `ROR RD, RN, #Int`

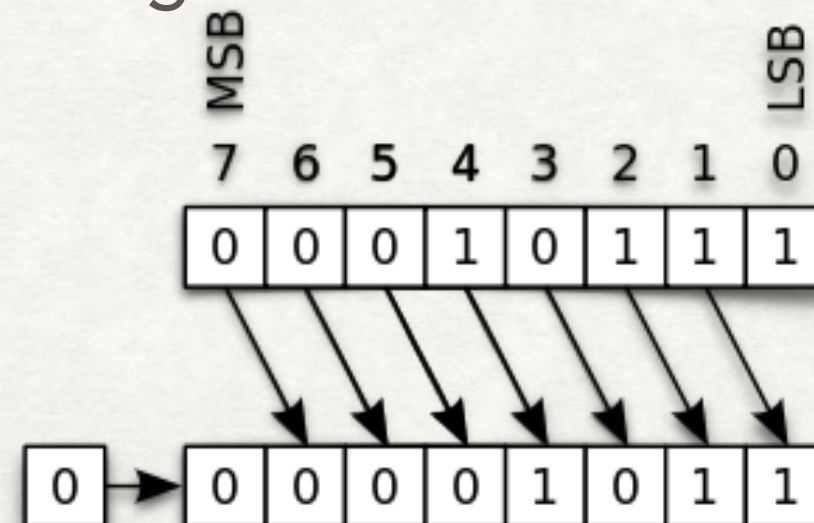
LINGUAGEM DE MÁQUINA

INSTRUÇÕES LÓGICAS: SHIFT LÓGICO

- **Shift Lógico à esquerda:** Desloca n bits da esquerda para a direita, adicionando novos zeros nos bits menos significativos. Os bits deslocados são descartados.



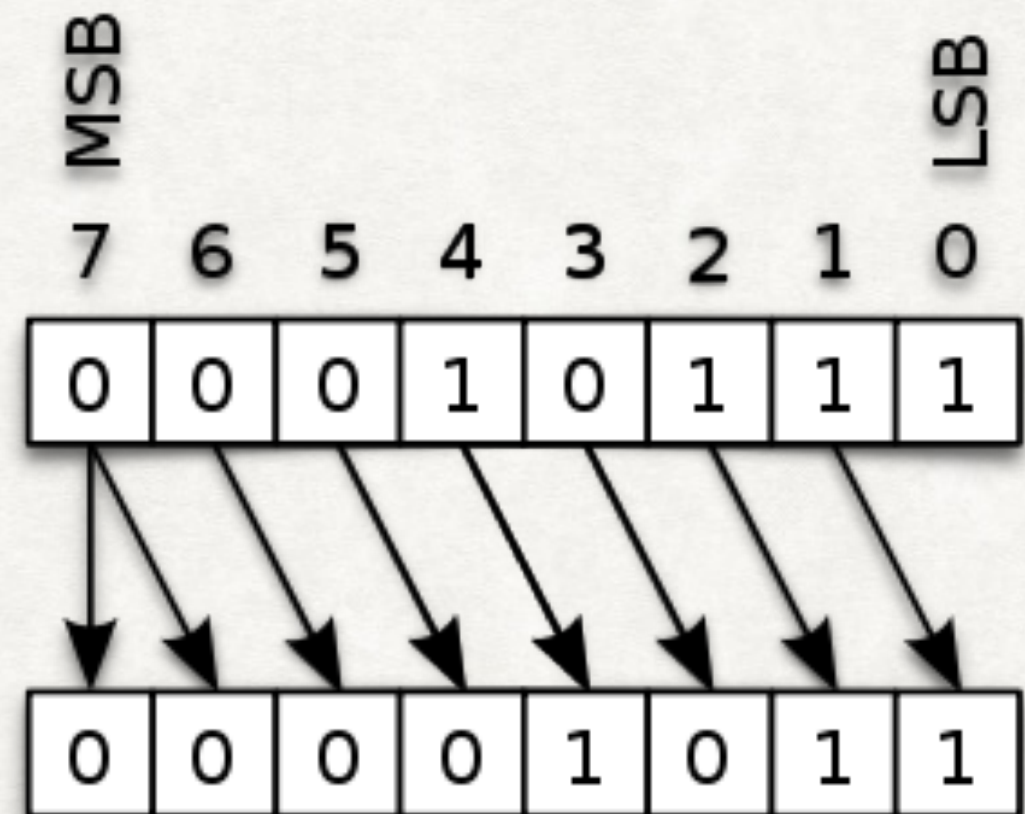
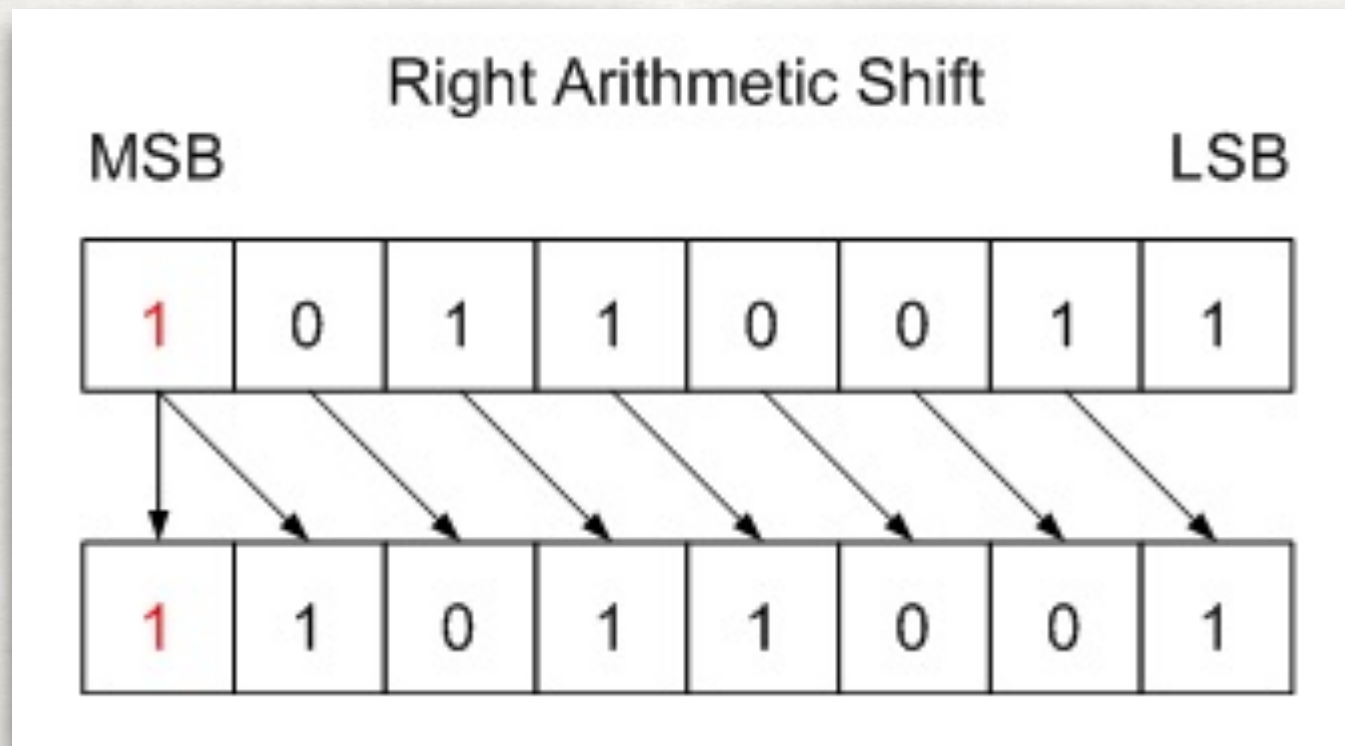
- **Shift Lógico à direita:** Desloca n bits da direita para a esquerda, adicionando novos zeros nos bits mais significativos. Os bits deslocados são descartados.



LINGUAGEM DE MÁQUINA

INSTRUÇÕES LÓGICAS: SHIFT LÓGICO

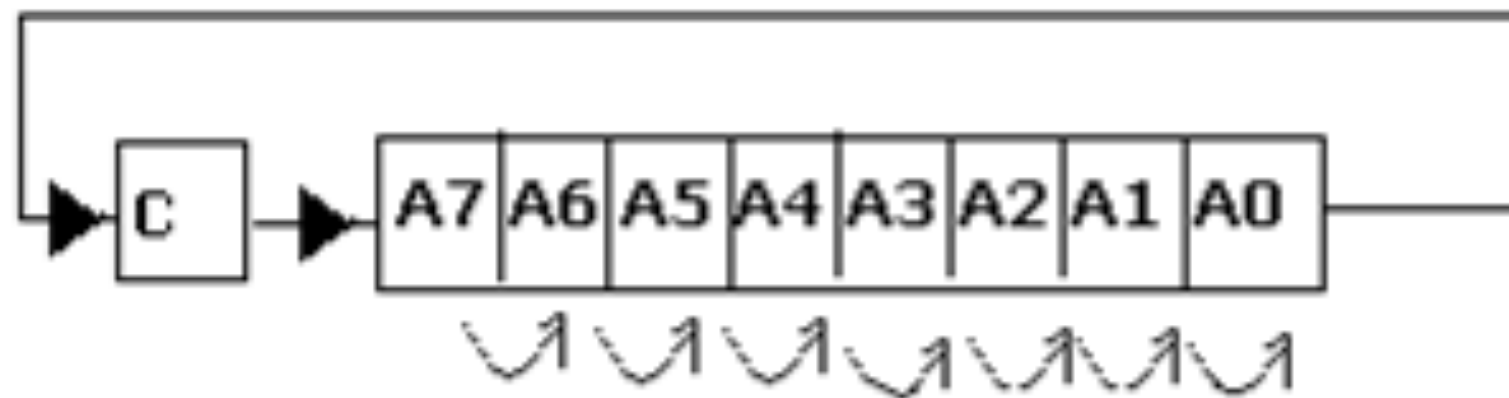
- **Shift Aritmético à direita:** Desloca n bits da direita para a esquerda, porém preenchendo os espaços vazios com o bit de sinal. Quando o bit de sinal é igual a zero, essa operação é igual a um Shift Lógico à direita.



LINGUAGEM DE MÁQUINA

INSTRUÇÕES LÓGICAS: ROTAÇÃO A DIREITA

- Rotação à direita: Guarda no bit de *carry* o bit menos significativo e empurra os bits *n* vezes para a direita, sem descartar qualquer bit. Essa instrução causa apenas uma reorganização dos bits.
- Essa operação é extremamente custosa pois a rotação é um processo linear.



Rotate Right through Carry (RRC)

LINGUAGEM DE MÁQUINA

INSTRUÇÕES LÓGICAS: COMPARE

- Para comparar valores armazenados em registradores, usamos a instrução:

CMP RN , RM

CMP RN , #INT

- Quando essa instrução é acionada, ocorre uma subtração dos parâmetros, e o resultado é analisado não levando em conta o valor, mas sim suas propriedades indicando igualdade; maior que; menor que;

LINGUAGEM DE MÁQUINA

INSTRUÇÕES LÓGICAS

- Ao realizar a instrução **CMP** os bits de controle das comparações são armazenados nos 4 primeiros bits do registrador 15 (PC). Ao fazer uma comparação, os bits são alterados gerando os seguintes resultados:


Suffix	Description	Flags tested
EQ	Equal	Z=1
NE	Not equal	Z=0
CS/HS	Unsigned higher or same	C=1
CC/LO	Unsigned lower	C=0
MI	Minus	N=1
PL	Positive or Zero	N=0
VS	Overflow	V=1
VC	No overflow	V=0
HI	Unsigned higher	C=1 & Z=0
LS	Unsigned lower or same	C=0 or Z=1
GE	Greater or equal	N=V
LT	Less than	N!=V
GT	Greater than	Z=0 & N=V
LE	Less than or equal	Z=1 or N!=V
AL	Always	

LINGUAGEM DE MÁQUINA

INSTRUÇÕES DE SALTO

- As instruções de salto (*Branch*) realizam alterações no **PC** (Program Counter), mudando a ordem de execução das instruções no código.
- Branch incondicional: **B** *alvo* (por exemplo: *go to*)
- Branch condicional: **BNE** *alvo* (por exemplo: *If then else*)
 - Saltos condicionais fazem uma checagem ao Registrador de Estado para analisar se a condição está satisfeita

	BNE	<i>else</i>
	ADD	<i>R1,R1,#1</i>
	B	<i>fim</i>
<i>else</i>	ADD	<i>R2,R2,#1</i>



LINGUAGEM DE MÁQUINA

INSTRUÇÕES DE SALTO

Branch	Interpretation	Normal uses
B	Unconditional	Always take this branch
BAL	Always	Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC	Carry clear	Arithmetic operation did not give carry-out
BLO	Lower	Unsigned comparison gave lower
BCS	Carry set	Arithmetic operation gave carry-out
BHS	Higher or same	Unsigned comparison gave higher or same
BVC	Overflow clear	Signed integer operation; no overflow occurred
BVS	Overflow set	Signed integer operation; overflow occurred
BGT	Greater than	Signed integer comparison gave greater than
BGE	Greater or equal	Signed integer comparison gave greater or equal
BLT	Less than	Signed integer comparison gave less than
BLE	Less or equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Unsigned comparison gave lower or same

LINGUAGEM DE MÁQUINA

INSTRUÇÕES DE SALTO

'C' Program fragment	ARM program using branching instructions
<pre> if (r0==0) { r1=r1+1; } else { r2=r2+1; } </pre>	<pre> CMP r0,#0 BNE else ADD r1,r1,#1 B end else ADD r2,r2,#1 end --- </pre> <p> Instructions - 5 Memory space - 20 bytes No. of cycles - 5 or 6 </p>

LINGUAGEM DE MÁQUINA

BONS MODOS DE PROGRAMAÇÃO ARM

- É possível carregar um valor para um registrador de maneira rápida por meio da instrução: `MOV RD, #Int`
- É importante separar a **região de dados**, para definir valores na memória, da **região de instruções**, que alteram valores dentro da CPU
- A diretiva `AREA_var` é usada para isso. O sugerido é o dividir da seguinte maneira:

```
AREA_Variaveis
ganhoDoDia      DCD      74
ganhoSemana     DCD      12,10,56,76
                ADR      R1, ganhoDoDia
                ADR      R7, ganhoSemana

AREA_Instrucoes

                MOV      R3,#10
                CMP      R3,#11
                BLT      menor
                BEQ      skip
                MOV      R2,#10

skip
menor           ADD      R3,R3,R3
```


LINGUAGEM DE MÁQUINA

AGORA É A SUA VEZ

- Faça um código em assembly que realize a seguinte tarefa:

```
n = 1
while(n < 10){
    n++;
}
```

LINGUAGEM DE MÁQUINA

AGORA É A SUA VEZ

- Faça um código em assembly que realize a seguinte tarefa:

```
n = 15;  
m = 1;  
while(n > 5){  
    if(m > n){  
        n--;  
    }  
    m++;  
}
```


LINGUAGEM DE MÁQUINA

INDICAÇÕES PARA O ESTUDO

- https://salmanarif.bitbucket.io/visual/supported_instructions.html (Instruções aceitas pelo VisUAL)
- <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0473f/Babbdajb.html> (Site da ARM explicando todas as instruções)
- <http://www.davespace.co.uk/arm/introduction-to-arm/> (Site explicando todos os conceitos de todas as instruções)
- <https://www.slideshare.net/PrDinesh1/arm-7-detailed-instruction-set>
- <http://slideplayer.com/slide/8423014/>
- <http://thinkingeek.com/arm-assembler-raspberry-pi/>

