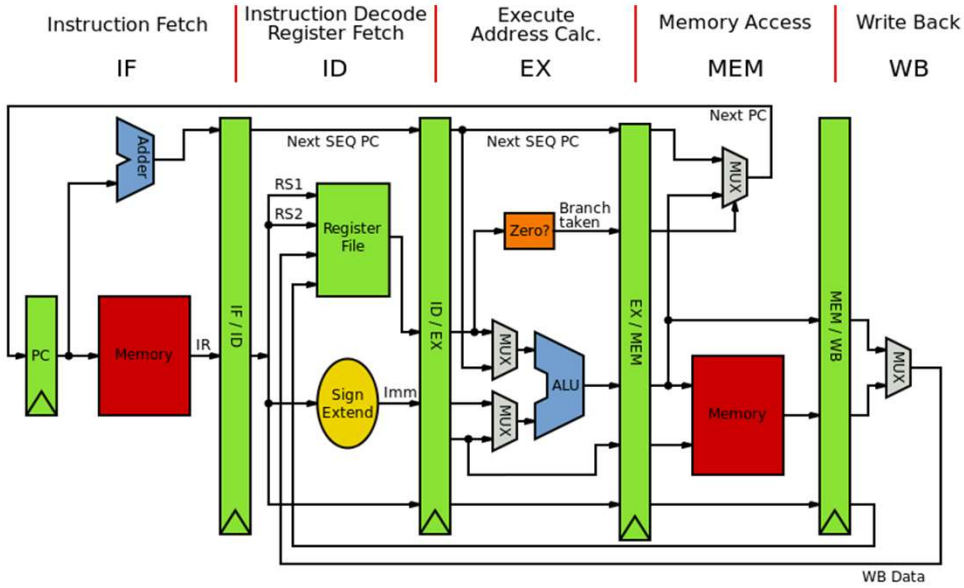
Breve Introdução a Arquitetura de Computadores

Marcelo Barros

Arquitetura de Computadores

A arquitetura de computadores é um conjunto de regras e métodos que descrevem a organização, funcionalidades e implementação de sistemas computacionais.



Referência: https://en.wikipedia.org/wiki/Computer_architecture

Arquitetura de Computadores

(Algumas) questões respondidas através de arquitetura de computadores:

- Quantos e quais são os registros presentes na plataforma ?
- De que forma o sistema é inicializado ?
- Como e onde as variáveis globais são alocadas ?
- De que forma é feito o gerenciamento da pilha e do stack ?
- Como os vetores de interrupções são usados ?
- Quais são os modos de endereçamento do microcontrolador ?
- Onde ficam as memórias de dados e de programa ? Como elas são acessadas ?

Conjunto de Instruções

Conjunto de instruções (tradução de *instruction set*) são as operações que um processador, microprocessador, microcontrolador, CPU ou outros periféricos programáveis suporta, fornece ou disponibiliza para o programador, ou seja, é a representação em mnemônicos do código de máquina, com a finalidade de facilitar o acesso ao componente.

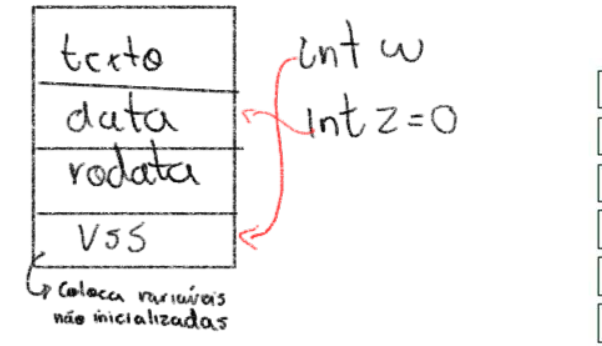
Referência: https://pt.wikipedia.org/wiki/Conjunto_de_instruções

Várias decisões são tomadas no projeto de uma arquitetura

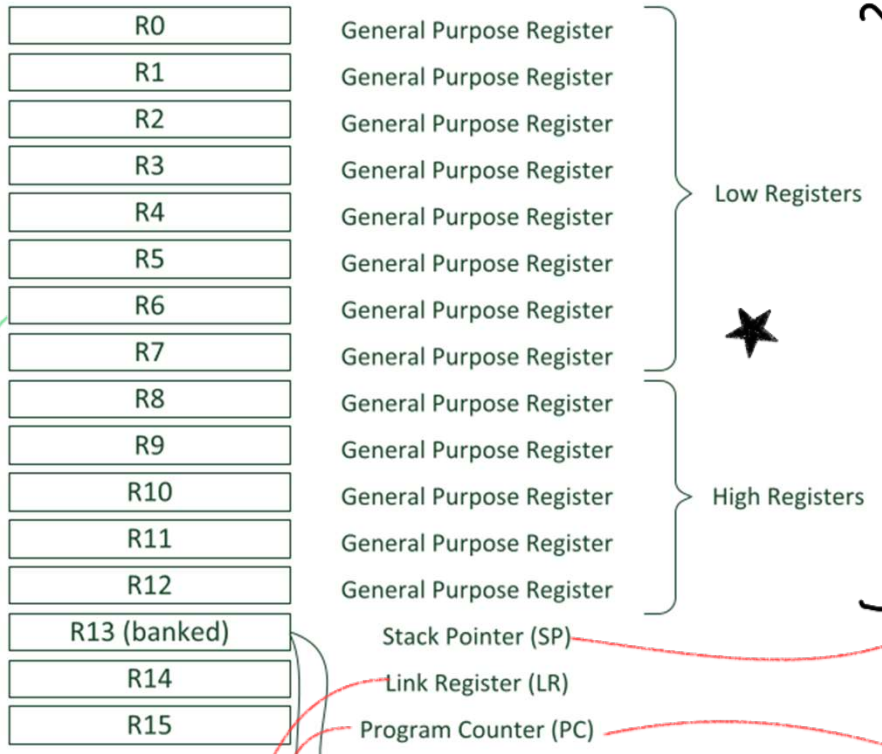
- Quantidade de registros
- Interligação entre elementos
- Registros genéricos x registros específicos
- Instruções complexas x instruções simples
- Instruções de tamanho variável ou não

Conjunto de Registros do Cortex M

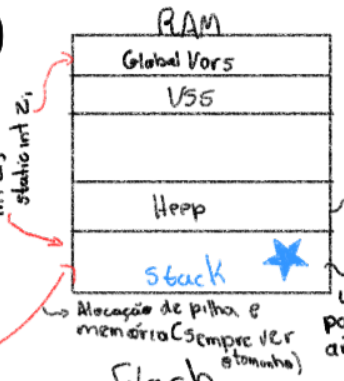
STM32
Cortex M4 (ARM)

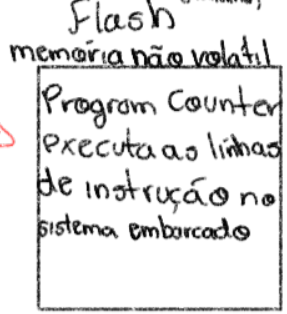


Registros: área do processador que recebe as instruções



Serve para qualquer coisa



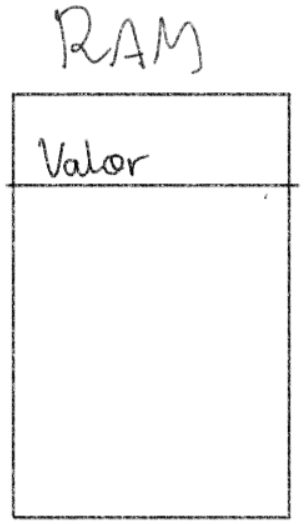


Controle de pilhas

Quando tem uma parada de sub rotina o (LR) aponta essa rotina no (PC).

Do C ao Assembly

- Como se traduz, para Assembly, o seguinte código?
  - int *r0 = (int *) 0x20;
  - int a = *(r0 + 0);
  - a = a + 1
  - *(r0 + 0) = a;
- Questões:
  - Onde são armazenados os valores ?
  - Como são codificados as instruções ?
  - Como é feita a busca e salvamento do valor da variável da memória ?



Cortex M
Adicionando

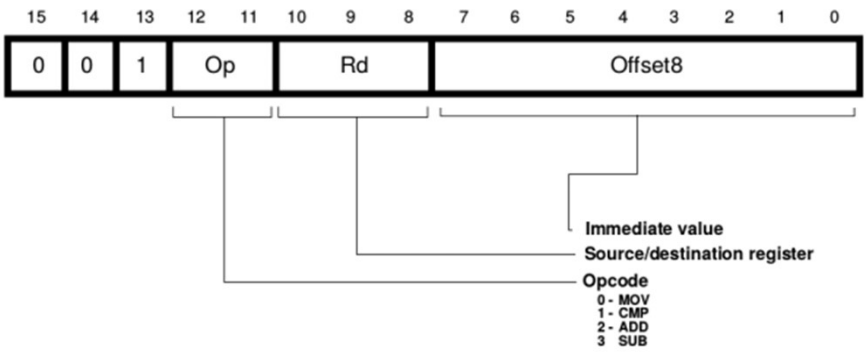
- `adds r1,#1  // a=a+1`
- `0011 0 001 00000001b => 3101h`

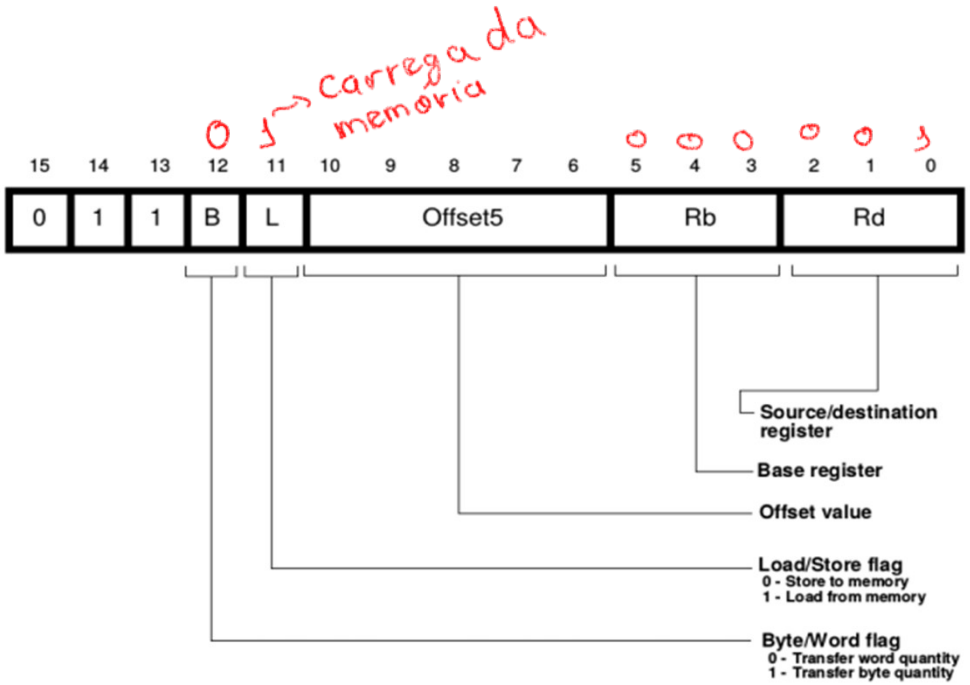
Cortex M
Carregando da memória

- Suponha que a variável esteja no endereço 0x20 da memória.
- Considere que a plataforma Cortex M não opera diretamente na memória (não é possível somar um diretamente na variável em RAM, com uma instrução)

Cortex M
Carregando da memória

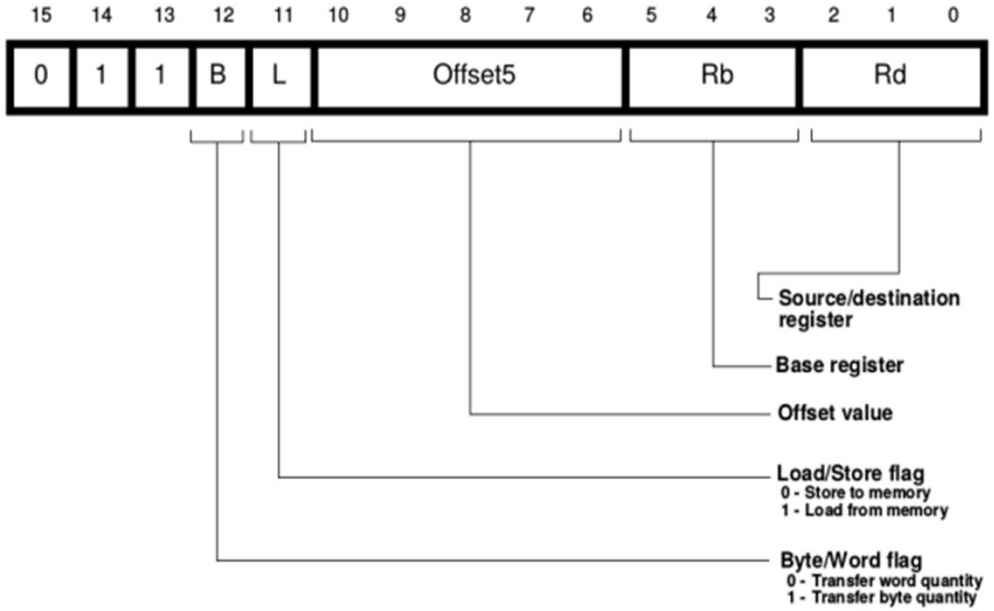
- `movs r0, #32` `// int *r0 = (int *) 0x20;`
- `ldr r1, [r0, #0]` `// int a = *(r0 + 0);`
- `0010000000100000b => 2020h`
- `0110100000000001b => 6801h`





Cortex M
Salvando na memória

- `str r1, [r0, #0]   // *(r0 + 0) = a;`
- `0110000000000001b => 6001h`



Cortex M
Programa Final

```
movs   r0, #32
ldr    r1, [r0, #0]
adds   r1, #1
str    r1, [r0, #0]
```

```
// a = a + 1
int *r0 = (int *) 0x20;
int a = *(r0 + 0);
a = a + 1
*(r0 + 0) = a;
```

Código de máquina: 2020h 6801h 3101h 6001h

Conjunto de Instruções

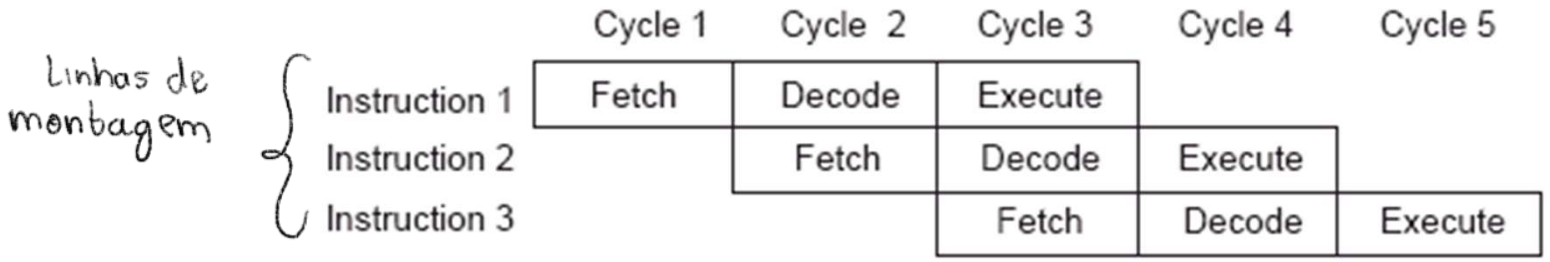
Instruções pequenas

- RISC (*Reduced Instruction Set Computer*): o hardware necessário para execução de instruções RISC é geralmente mais simples, com decodificação do código de máquina sendo feita comumente em um ciclo de clock.
- CISC (*Complex Instruction Set Computer*): nestas plataformas, as instruções realizam tarefas mais complexas e são geralmente decodificadas através de um microcódigo, ou seja, de conjuntos de instruções internas, realizadas pelo processador. Isto permite que a geração do código de máquina seja mais simples e os programas executáveis menores mas gera processadores bastante complexos.

Instruções grandes (SAMO)

Pipeline

- O conceito de pipeline está relacionado com a divisão do processador em unidades independentes que operam em paralelo.
- No Cortex M, a divisão do processador em unidades de busca, decodificação e execução da instrução, permite que isso seja feito em paralelo.



Modos de Endereçamento

- Cada instrução do processador precisa definir como os seus argumentos (operandos) são carregados.
- A forma de tratar os operandos é conhecida como modo de endereçamento e varia em complexidade e quantidade entre plataformas diferentes.

Modos de Endereçamento

- **Endereçamento Imediato**: o operando está codificado junto com o opcode (código de máquina), sendo carregado da memória juntamente com a instrução.
  - movs r0, #32
- **Endereçamento Direto**: permite que uma posição de memória seja usada como parte da instrução, manipulada diretamente pelo microcontrolador. Este modo não existe na plataform Cortex M.
- **Endereçamento de Registrador**: similar ao modo direto mas, ao invés de endereços de memória, usa registros do microcontrolador. Operações direto por registros
  - muls r1,r2,r3

Modos de Endereçamento

- **Endereçamento Indireto de Registrador**: assemelha-se ao endereçamento direto mas com a diferença de ter o endereço de memória em um registro, usado de forma indireta, ao invés de constituir o opcode.
  - ldr r1, [r0]
- **Endereçamento Indexado**: muito similar ao modo endereçamento indireto de registrador mas com a possibilidade de adicionar um deslocamento constante.
  - str r1, [r0,#10]

Modos de Endereçamento

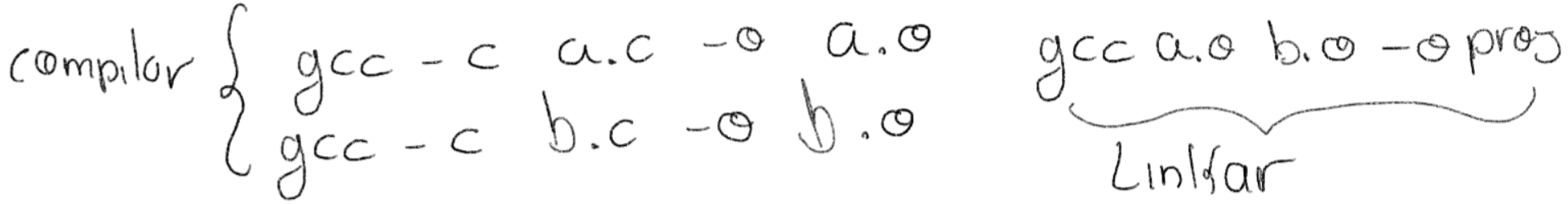
- **Endereçamento de Pilha e Endereçamento por Instruções de Desvio**: existem situações onde é desejável o uso de um registro base diferente de um registro geral, como acontece no modo de endereçamento indexado. Na plataforma Arm Cortex, push e pop são bons exemplos.
  - push { r1 } <==> str r1, [sp, #-4]!
  - push { r1, r2, r3, r4 }

Contador de Programa ou Program Counter (PC)

- O PC é o registro que indica qual instrução está sendo (ou vai ser) executada. É um ponteiro para a memória de programa, sendo atualizado a cada nova instrução executada.
- Quando existem desvios no programa, é o PC que é alterado. Uma chamada de sub-rotina, por exemplo, passa necessariamente por uma mudança do valor do PC, que passa a apontar para outra região da memória de programa, no caso, para a região onde está localizada a sub-rotina.
- Ao ser inicializado, o processador carrega o PC com o endereço de partida para a arquitetura em questão. Então, passa a buscar as instruções e executá-las.
- Na arquitetura Arm Cortex M o PC é representado pelo registro R15, sendo específico para este fim.

Ponteiro de Pilha ou Stack Pointer (SP)

- O SP tem a finalidade de controlar o uso da pilha, permitindo a alocação de variáveis temporárias na pilha assim como a passagem de parâmetros na chamada de função.
- O SP é representado pelo R13 na arquitetura Arm Cortex.
- Este comportamento não é exclusivo do Arm, é também usado em outras arquiteturas também, sendo a definição de uma área de stack sempre uma tarefa importante e necessária se você pretende programar em C.



Ponteiro de Pilha ou Stack Pointer (SP)

- Quando se programa em Assembly diretamente você tem o controle total do SP. Pode até mesmo não usá-lo, caso trate adequadamente os registros gerais. No entanto, quando se tem um compilador envolvido, existem convenções adotadas para o uso do SP.
- O padrão usado atualmente para Arm Cortex é conhecido como EABI (Embedded ABI):
  - Registros de R0 a R3 são usados para passagem de argumentos para funções de até quatro parâmetros.
  - Funções com mais de quatro parâmetros devem fazer uso da pilha para esta passagem.
  - O retorno, por sua vez, é via registro R0.
- O padrão EABI permite a interoperação entre ferramentas de fabricantes diferentes.

EABI e Stack Pointer

```
int a = 10;

int __attribute__ ((noinline))
soma5(int v1,int v2,int v3, int v4,int v5)
{
  return (v1 + v2 + v3 + v4 + v5);
}

int main(void)
{
  int v1, v2, v3, v4, v5;
  v1 = 1; v2 = 2;
  v3 = 3; v4 = 4;
  v5 = 5;
  a = soma5(v1,v2,v3,v4,v5);
  return a;
}
```

```
080001d8 <soma5>:
080001d8: 4401 add r1, r0
080001da: 440a add r2, r1
080001dc: 9800 ldr r0, [sp, #0]
080001de: 4413 add r3, r2
080001e0: 4418 add r0, r3
080001e2: 4770 bx  lr
080001e4 <main>:
080001e4: b507 push {r0, r1, r2, lr}
080001e6: 2305 movs r3, #5
080001e8: 9300 str  r3, [sp, #0]
080001ea: 2102 movs r1, #2
080001ec: 2203 movs r2, #3
080001ee: 2304 movs r3, #4
080001f0: 2001 movs r0, #1
080001f2: f7ff fff1 bl 80001d8 <soma5>
080001f6: 4b02 ldr r3, [pc, #8];(8000200<main+0x1c>)
080001f8: 6018 str r0, [r3, #0]
080001fa: b003 add sp, #12
080001fc: f85d fb04 ldr.w   pc, [sp], #4
08000200: 20000000 .word   0x20000000
...
```

Chamada da função

Registro de Status ou Status Register (SR)

Observa o status da ultima operação do processador

Controle de fluxo por exemplo em loops a -> 0

O SR é geralmente composto por flags que podem ser modificados a cada instrução executada. Existem flags para várias condições. Por exemplo:

- estouro de contagem (overflow) -> Indicações de status (saturação)
- indicar se o resultado da operação realizada foi zero ou não,
- alertar quando um valor se torna negativo
- sinalizar se as interrupções estão ligadas ou não
- evidenciar quando uma operação de vai um aconteceu (carry)

- A quantidade de flags e o número de registro de status necessários para mapear todos os flags é altamente dependente da plataforma. No Cortex, o registro de status é, na realidade, uma composição de três registros diferentes, com funções adicionais relacionadas ao tratamento de exceções.

	31	*30*	*29*	*28*	*27*	*26:25*	*24*	*23:20*	*19:16*	*15:10*	*9*	*8*	*7*	*6*	*5*	*4:0*
APSR	N	Z	C	V	Q				GE*							
IPSR												Exception Number				
EPSR						ICI/IT	T			ICI/IT						

$x = x + y$
$x = y - x$
$y = x - y$

Registro de Status ou Status Register (SR)

- Uma situação corriqueira do uso de flags do registro de status são laços e testes de condições:

```
int main(void)
{
  volatile int a = 10;

  while(a >= 0)
    a--;
  return 0;
}
```

```
080001d8 <main>:
080001d8: b082 sub    sp, #8
080001da: 230a movs   r3, #10
080001dc: 9301 str    r3, [sp, #4]
080001de: 9b01 ldr    r3, [sp, #4]
080001e0: 2b00 cmp    r3, #0
080001e2: dd02 ble.n  80001ea <main+0x12>
080001e4: 9b01 ldr    r3, [sp, #4]
080001e6: 3b01 subs   r3, #1
080001e8: e7f8 b.n    80001dc <main+0x4>
080001ea: 2000 movs   r0, #0
080001ec: b002 add    sp, #8
080001ee: 4770 bx     lr
```

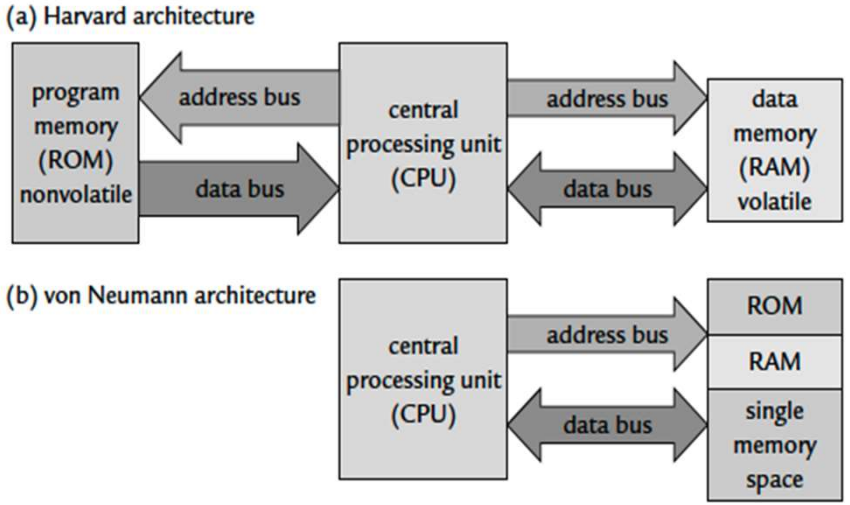
Registro de status

N: 1 se negativo
Z: 1 se zero
C: 1 se teve carry
V: 1 se overflow

Von Neumann x Harvard

Acesso à memória

- Um aspecto interno de arquiteturas de computadores diz respeito a como a unidade de processamento faz acesso aos dados e ao programa:
  - Quando este acesso é feito através de um mesmo barramento, esta arquitetura é conhecida como Von Neumann.
  - Se a memória dos dados e de programa utilizam barramentos dedicados, a arquitetura é conhecida como Harvard.
- Harvard permite um maior paralelismo e o uso de caches separados para dados e código



DMA (Direct Memory Access)

- Permite que periféricos acessem a memória diretamente, não passando pela CPU
- Libera a CPU para outras atividades
- Tipicamente:
  - Uma operação de DMA é programada entre periférico e memória
  - A operação é executada e uma interrupção ocorre para avisar a CPU
- Operações comuns:
  - Memória → memória
  - Device → memória
  - Memória → device

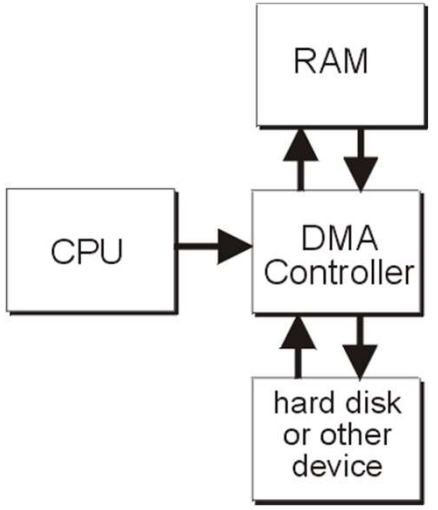


Imagem: http://is.gd/ylS6DG

Outros Exemplos: AVR

- Originalmente concebida por dois estudantes no Norwegian Institute of Technology (NTH), Alf-Egil Bogen e Vegard Wollan.
- Vendida para a Atmel posteriormente na década de 90 (agora Microchip, 2016)
- Microcontrolador de 8 bits
- Inovadora, quando lançada:
  - Harvard
  - Flash interna
  - RISC
  - Oscilador interno
  - EEPROM
- O AT90S1200 foi o primeiro controlador comercial da linha (1997)

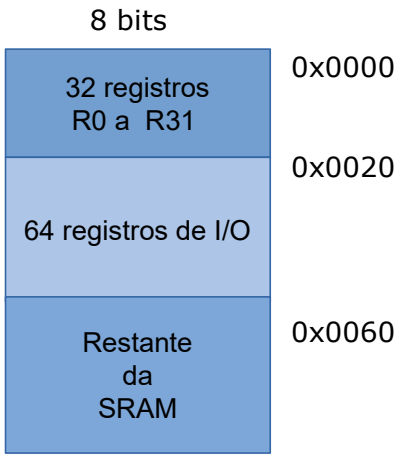
Principais Membros da Família AVR

- TinyAVR: mais simples, até 32kB de flash, até 32 pinos, menos periféricos
- MegaAVR: até 256kB de flash, até 100 pinos, multiplicação por hardware, mais periféricos
- XMEGA: até 384kB, DMA, até 100 pinos, muitos periféricos
- AVR32: os mais recentes, mas agora baseados em Cortex M

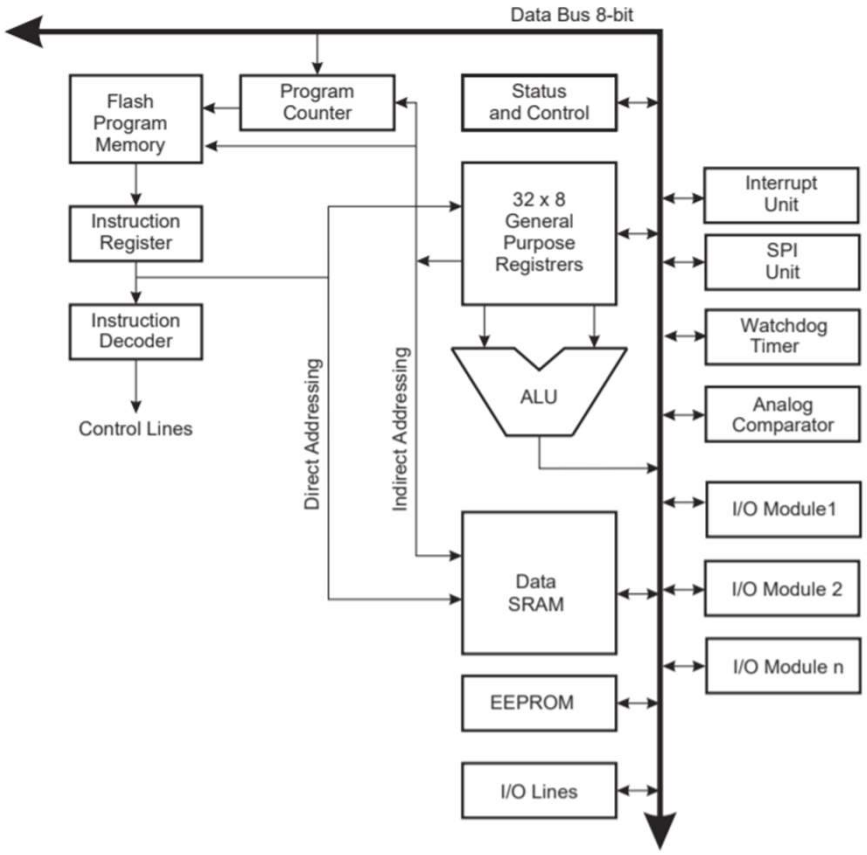
Fonte: https://en.wikipedia.org/wiki/AVR_microcontrollers

Arquitetura ATmega328P

- RISC
- 32 registros gerais de 8 bits
- 64 registros de I/O
- PC de 16 bits
- Uma instrução por ciclo (20MHz máximo)
- Multiplicação em hardware (2 ciclos)
- 4 a 32 kB de flash, até 2kB de RAM, até 1kB de EEPROM
- Stack pointer mapeado na SRAM

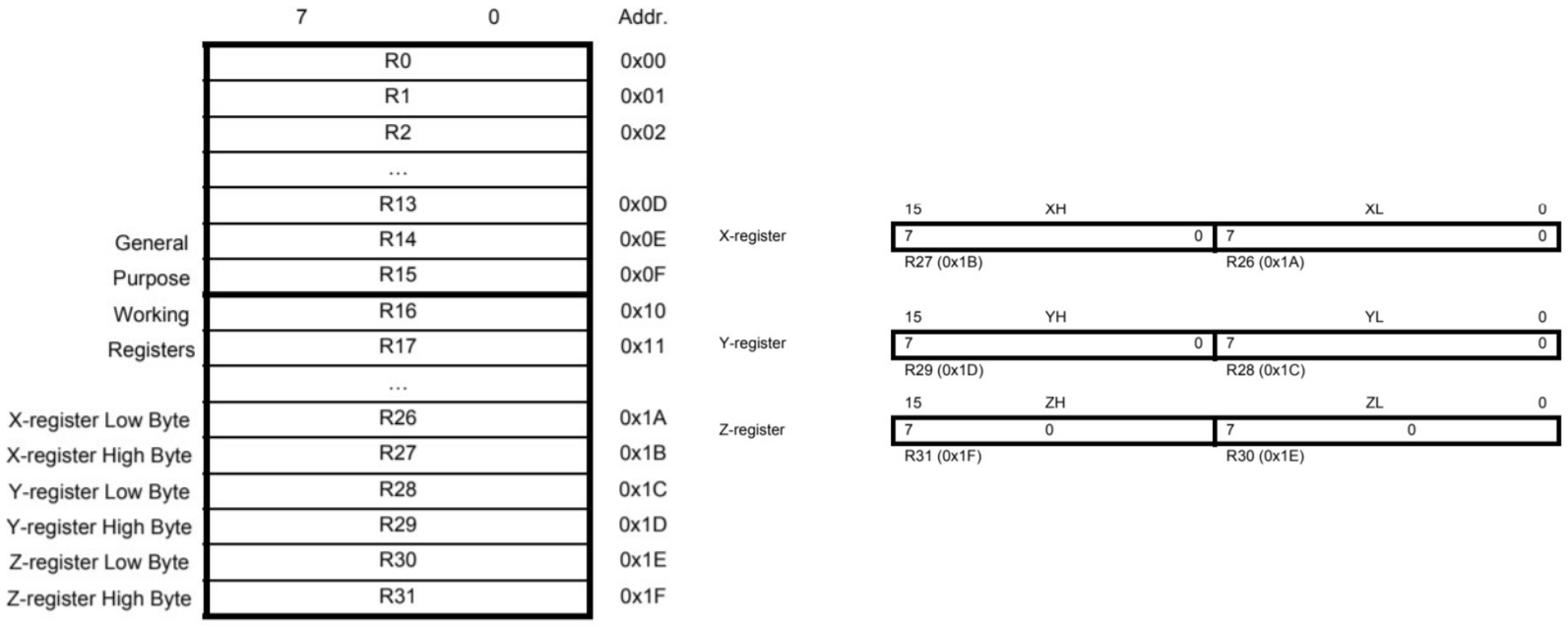


Arquitetura ATmega328P



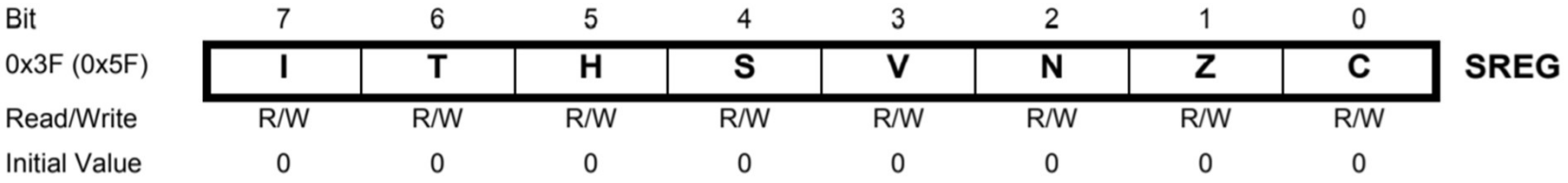
Arquitetura ATmega328P

Detalhamento de uso dos registros. Os últimos seis podem operar em conjunto para endereçamento em 16 bits à memória.



Arquitetura ATmega328P

- Registro de status, principais flags:
  - I: flag de habilitação global de interrupção
  - V: flag de overflow em operações de complemento de 2
  - N: flag de negativo
  - Z: flag de zero
  - C: flag de carry



Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Arquitetura ATmega328P

Bit	15	14	13	12	11	10	9	8	
0x3E (0x5E)	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
0x3D (0x5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	
	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	

Table 7-1. Stack Pointer instructions

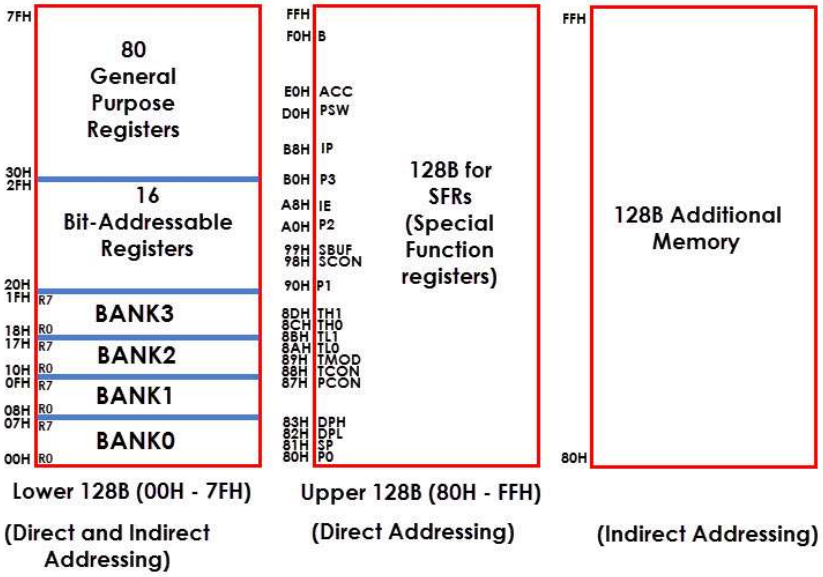
Instruction	Stack pointer	Description
PUSH	Decremented by 1	Data is pushed onto the stack
CALL ICALL RCALL	Decremented by 2	Return address is pushed onto the stack with a subroutine call or interrupt
POP	Incremented by 1	Data is popped from the stack
RET RETI	Incremented by 2	Return address is popped from the stack with return from subroutine or return from interrupt

Outro Exemplo: 8051

- Desenvolvido pela Intel, década de 80
- CISC, 8 bits
- Ainda em uso, na forma de core 8051 em diversos chips
- Mapa de registros relativamente complexo (de entender):
  - Registro ACC, B, PORT
  - Lower 128 RAM bytes
    - Primeiros 32 registros divididos em 4 bancos que só podem ser acessados em grupos de 8
    - Depois, existem duas áreas (área de bits e RAM de uso geral)
  - Upper 128 RAM bytes
    - Acesso indireto somente, uso geral (0x80 a 0xFF)

Registros 8051

- ACC (acumulador)
- B (operações matemática)
- PSW (aproximadamente um registro de status)
- SP Stack pointer

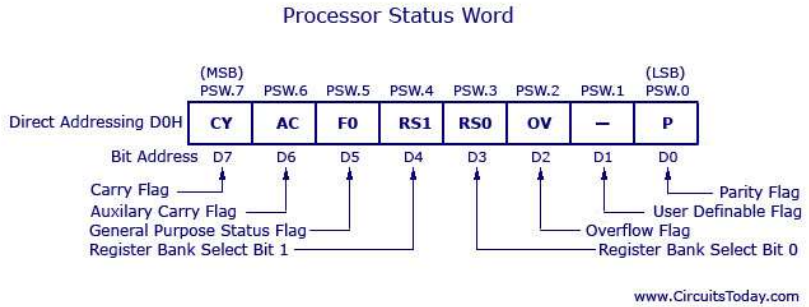


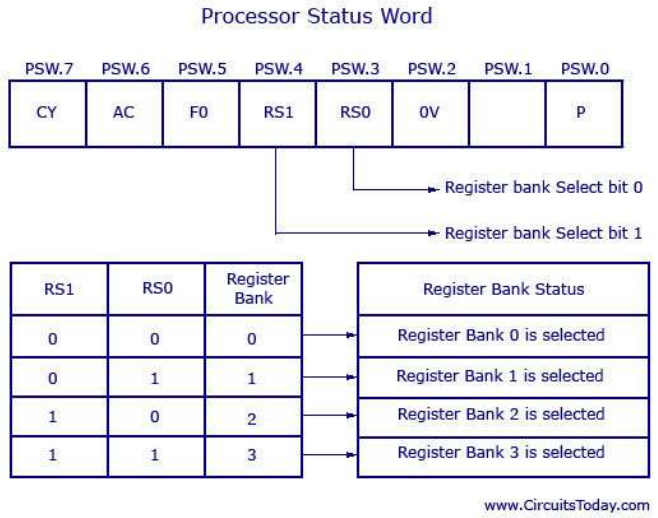
Name of the Register	*Function*	*Internal RAM Address (HEX)*
ACC	Accumulator	E0H
B	B Register (for Arithmetic)	F0H
DPH	Addressing External Memory	83H
DPL	Addressing External Memory	82H
IE	Interrupt Enable Control	A8H
IP	Interrupt Priority	B8H
P0	PORT 0 Latch	80H
P1	PORT 1 Latch	90H
P2	PORT 2 Latch	A0H
P3	PORT 3 Latch	B0H
PCON	Power Control	87H
PSW	Program Status Word	D0H
SCON	Serial Port Control	98H
SBUF	Serial Port Data Buffer	99H
SP	Stack Pointer	81H
TMOD	Timer / Counter Mode Control	89H
TCON	Timer / Counter Control	88H
TL0	Timer 0 LOW Byte	8AH
TH0	Timer 0 HIGH Byte	8CH
TL1	Timer 1 LOW Byte	8BH
TH1	Timer 1 HIGH Byte	8DH

Fonte: https://www.electronicshub.org/8051-microcontroller-special-function-registers/

Registros 8051: PSW e Banco de Registros

- SP inicia apontando em 0x07 e incrementa ao ser usado (push)
- Logo, SP aponta no banco 1 e cresce
- Data Pointer (DPL e DPH) são usados para acesso a RAM além do endereço 0xFF
- PC de 16 bits

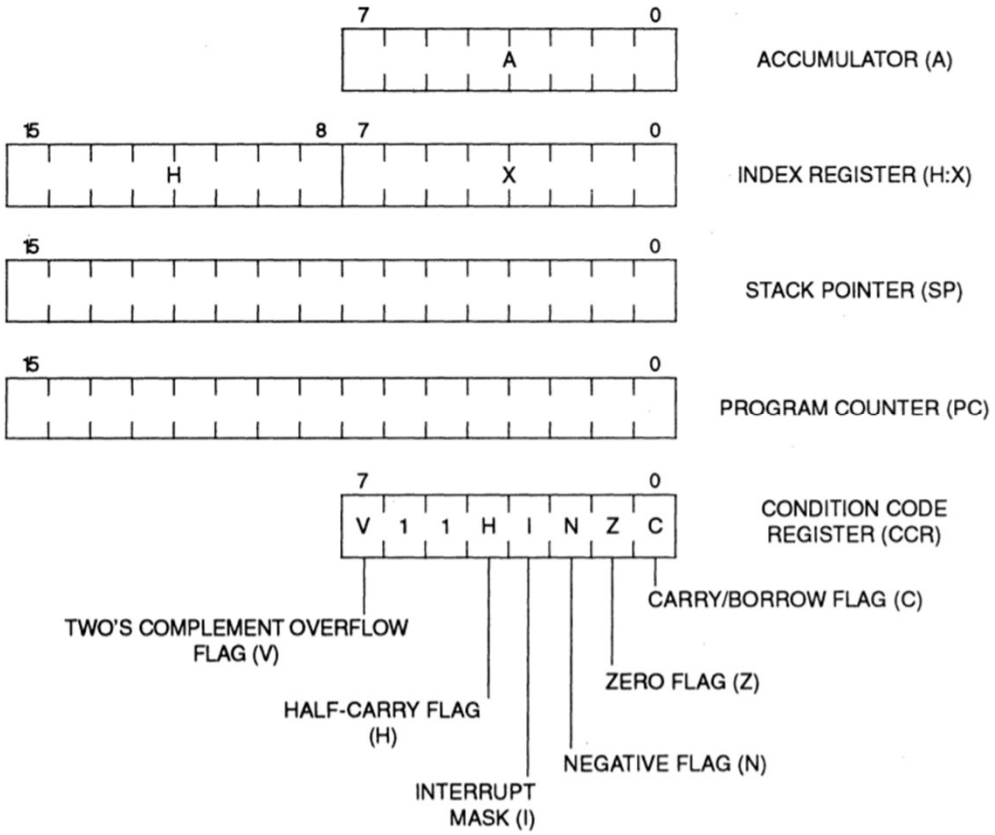




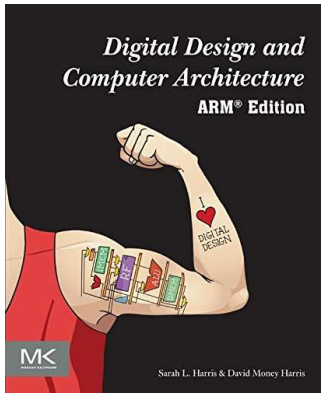
Motorola 68HC08

- A linha 6800 foi introduzida na década de 70
- CISC, 8 bits
- 64kB de endereçamento. Memória adicional através de paginação.
- 8MHz de clock
- PC e SP de 16 bits

Motorola 68HC08



Referências



Digital Design and Computer Architecture: ARM Edition
Editora Morgan Kaufmann
Sarah Harris, David Harris

Organização Estruturada de Computadores
Editora Pearson
Andrew S. Tanenbaum

Arquitetura de Computadores

(Algumas) questões respondidas através de arquitetura de computadores:

- Quantos e quais são os registros presentes na plataforma ?
- De que forma o sistema é inicializado ?
- Como e onde as variáveis globais são alocadas ?
- De que forma é feito o gerenciamento da pilha e do stack ?
- Como os vetores de interrupções são usados ?
- Quais são os modos de endereçamento do microcontrolador ?
- Onde ficam as memórias de dados e de programa ? Como elas são acessadas ?