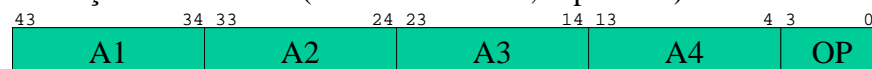


Arquiteturas de 4, 3, 2, 1 e 0 endereços

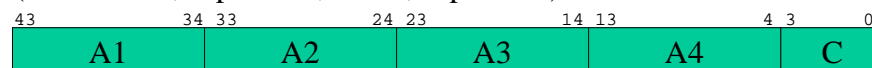
EDVAC 4 endereços - instruções de 44 bits

Instruções aritméticas ($A3 \leftarrow A1 \text{ OP } A2$; ir para A4)



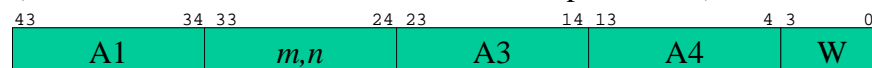
Instruções condicionais

(Se $A1 \geq A2$, ir para A3; senão, ir para A4)



Instruções de entrada/saída

(m : 1 = saída, 2 = entrada; n = número do “periférico”)



(transfere de/para palavras A1 .. A3; depois, ir para A4)

EDVAC principais inconvenientes

- memória de 1024 palavras de 44 bits
- cada instrução ocupa 44 bits na memória
- requer um acesso à memória para ler uma instrução
- requer vários acessos adicionais à memória para ler / escrever os operandos da instrução

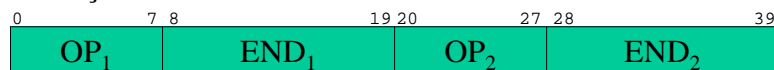
IAS 1 endereço - instruções de 20 bits

Formato de uma instrução



- memória de 4096 (2^{12}) palavras de 40 bits
- duas instruções por palavra de memória

Instruções na memória



IAS vantagens em relação ao EDVAC

- memória de 4096 palavras de 40 bits
- cada instrução ocupa 20 bits na memória; 2 instruções por palavra
- requer apenas um acesso à memória para ler duas instruções
- requer somente um acesso adicional à memória para ler / escrever o operando da instrução

IAS - desvantagens

- precisa de um registrador contador (PC) para indicar onde está na memória a próxima instrução a ser executada
- requer duas instruções para “mover” (copiar) dados na memória
- requer mais instruções para executar operações que envolvam mais de um operando na memória

Exemplo: operações aritméticas

Atribuir o valor de uma expressão aritmética a uma variável (posição de memória):

$$A \leftarrow ((B + C) * D + E - F) / (G * H)$$

Arquitetura com 4 endereços

OP	E1	E2	E3	E4
----	----	----	----	----

Endereço	Instrução	Comentário
e ₁	ADD B C A e ₂	Soma B com C, resultado em A; vai para e ₂
e ₂	MUL A D A e ₃	Multiplca A por D, resultado em A; vai para e ₃
e ₃	ADD A E A e ₄	Soma A com E, resultado em A; vai para e ₄
e ₄	SUB A F A e ₅	Subtrai F de A, resultado em A; vai para e ₅
e ₅	DIV A G A e ₆	Divide A por G, resultado em A; vai para e ₆
e ₆	DIV A H A e ₇	Divide A por H, resultado final em A; vai para e ₇
e ₇	HALT	Fim do programa

Arquitetura com 3 endereços

OP	E1	E2	E3
----	----	----	----

Endereço	Instrução	Comentário
e_1	ADD B C A	Soma B com C, resultado em A; incrementa PC
e_1+1	MUL A D A	Multiplica A por D, resultado em A; incrementa PC
e_1+2	ADD A E A	Soma A com E, resultado em A; incrementa PC
e_1+3	SUB A F A	Subtrai F de A, resultado em A; incrementa PC
e_1+4	DIV A G A	Divide A por G, resultado em A; incrementa PC
e_1+5	DIV A H A	Divide A por H, resultado final em A; incrementa PC
e_1+6	HALT	Fim do programa

- requer uso de contador de programa (PC)
- instruções de desvio condicional e incondicional

Arquitetura com 2 endereços

OP	E1	E2	$E1 \leftarrow (E1 \text{ OP } E2)$
----	----	----	-------------------------------------

Endereço	Instrução	Comentário
e_1	MOV A B	Move B para A
e_1+1	ADD A C	Soma A com C, resultado em A
e_1+2	MUL A D	Multiplica A por D, resultado em A
e_1+3	ADD A E	Soma A com E, resultado em A
e_1+4	SUB A F	Subtrai F de A, resultado em A
e_1+5	DIV A G	Divide A por G, resultado em A
e_1+6	DIV A H	Divide A por H, resultado final em A
e_1+7	HALT	Fim do programa

- resultado substitui valor de um dos operandos
- nova necessidade: instruções de movimentação (cópia)

Arquitetura com 1 endereço

OP

E1

Acum \leftarrow (Acum OP E1)

Endereço	Instrução	Comentário
e ₁	LDA B	Move B para Acumulador
e ₁ +1	ADD C	Soma <u>Acum.</u> com C, resultado no Acumulador
e ₁ +2	MUL D	Multiplica <u>Acum.</u> por D, resultado no Acumulador
e ₁ +3	ADD E	Soma <u>Acum.</u> com E, resultado no Acumulador
e ₁ +4	SUB F	Subtrai F do <u>Acum.</u> , resultado no Acumulador
e ₁ +5	DIV G	Divide <u>Acum.</u> por G, resultado no Acumulador
e ₁ +6	DIV H	Divide <u>Acum.</u> por H, resultado no Acumulador
e ₁ +7	STA A	Armazena <u>Acum.</u> no endereço de A
e ₁ +8	HALT	Fim do programa

- requer acumulador na unidade operacional
- requer instruções de LOAD / STORE para mover (copiar) dados na memória

Arquitetura com 0 endereços

OP

Notação (RPN): H G F E D C B + * + - / /

- todas operações são executadas sobre os dados que estão no topo da pilha (uma ou mais palavras, de acordo com a operação, são “retiradas” da pilha) e os resultados são colocados (“inseridos”) no topo da pilha
- requer capacidade de acessar pilha de dados (na memória ou em um banco de registradores especialmente para esta função), com registrador que aponta o topo da pilha a cada momento
- requer instruções de PUSH *end* / POP *end* (exceções - possuem 1 endereço) para mover (copiar) dados entre a memória e o topo da pilha
- estas arquiteturas não são implementadas isoladamente

Arquitetura com 0 endereços

OP

Endereço	Instrução	Comentário
e_1	PUSH H	Coloca H no topo (atual) da pilha
e_1+1	PUSH G	Coloca G no topo da pilha
e_1+2	PUSH F	Coloca F no topo da pilha
e_1+3	PUSH E	Coloca E no topo da pilha
e_1+4	PUSH D	Coloca D no topo da pilha
e_1+5	PUSH C	Coloca C no topo da pilha
e_1+6	PUSH B	Coloca B no topo da pilha
e_1+7	ADD	Topo da pilha recebe $B+C$ (E e C são retirados da pilha)
e_1+8	MUL	Topo recebe $(B+C)*D$
e_1+9	ADD	Topo recebe $(B+C)*D + E$
e_1+10	SUB	Topo recebe $(B+C)*D + E - F$
e_1+11	DIV	Topo recebe $((B+C)*D + E - F)/G$
e_1+12	DIV	Topo recebe $((B+C)*D + E - F)/G*H$
e_1+13	POP A	Topo da pilha é armazenado em A
e_1+14	HALT	Fim do programa