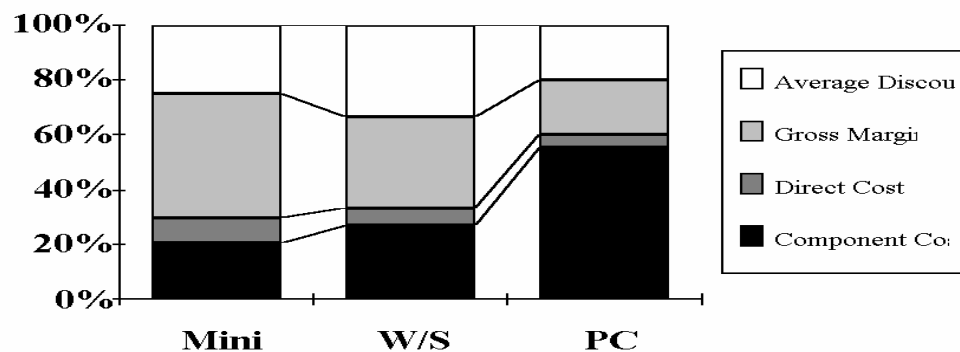




## Aula 04: Arquitetura do Conjunto de Instruções



## Revisão do Último Capítulo Preço vs. Custo





## Impacto da Redução das Margens de Lucros

- Economia?
- Laboratórios de pesquisa?
- Futuros produtos?
- Seus empregos?



## *Instruction Set Architecture*

- Visão geral do capítulo
  - Classificação do conjunto de instruções
  - Observar como aplicações utilizam ISA
  - Examinar ISA de um processador moderno (DLX)
  - Medições de utilização de ISA em computadores reais
- Mas você não vai projetar um conjunto de instruções...
  - Projetos só aparecem uma vez por década;
  - Compatibilidade em nível de código objeto é mais importante do que se pensava;
  - ISA modifica pouco entre computadores hoje em dia;
  - Oportunidades para novas ISA em ASICs (Application Specific IC) e arquiteturas reconfiguráveis;



## Instruction Set Architecture

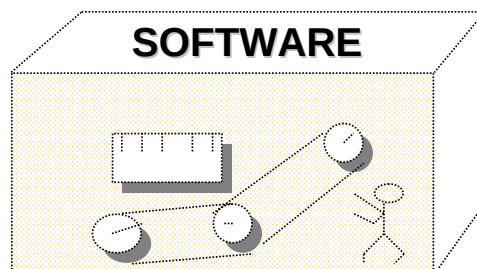
- **Anos 50s aos 60s:**  
Arquitetura de computadores  $\leftrightarrow$  aritmética de computadores
- **Anos 70 a metade dos 80s:**  
Arquitetura de computadores  $\leftrightarrow$  projeto do conjunto de instruções, especialmente ISA apropriado para compiladores
- **Anos 90s:**  
Arquitetura de computadores  $\leftrightarrow$  projeto de CPU, subsistema de memória, subsistema de I/O, multiprocessadores



## Arquitetura de computadores

*“os atributos de um sistema de computação na visão do programador, i.e., a estrutura conceitual e o comportamento funcional, ... em oposição à implementação física.”*

Amdahl, Blaaw, and Brooks, 1964





# Avaliação de ISA e sua Organização

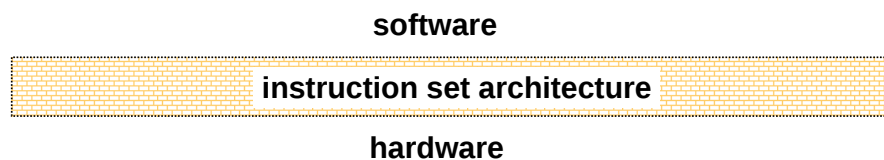


Conjunto de instruções: parte do processador visível ao programador ou pessoa que desenvolverá compiladores (o que você enxerga do hardware)



## Instruction Set Architecture

- The instruction set architecture serves as the interface between software and hardware
- It provides the mechanism by which the software tells the hardware what should be done
- Architecture definition:  
*"the architecture of a system/processor is (a minimal description of) its behavior as observed by its immediate users"*

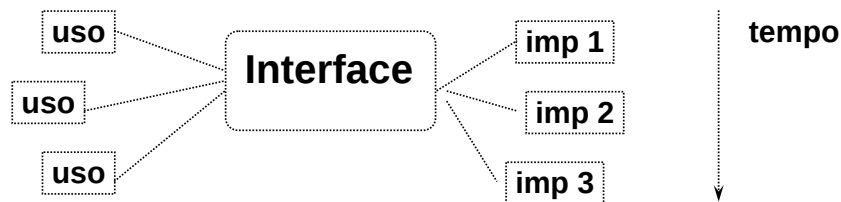




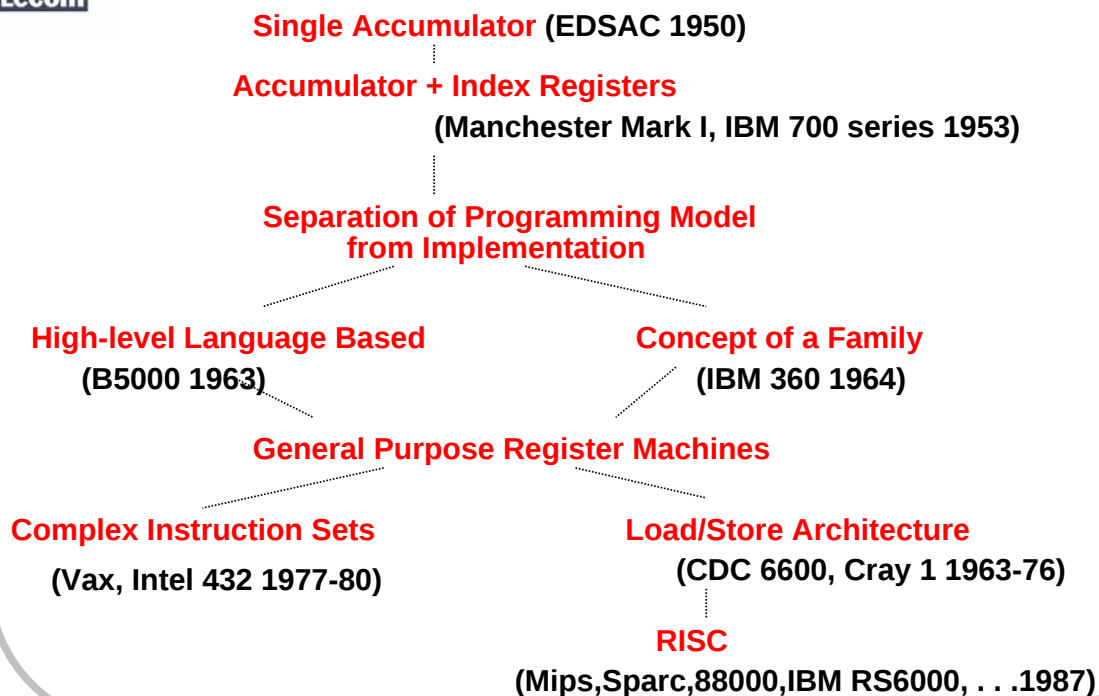
## Projeto de Interfaces

### Uma boa interface:

- Dura diversas implementações (portabilidade, compatibilidade)
- Usada em diferentes maneiras (generalidade)
- Provê funcionalidade conveniente aos níveis superiores
- Permite uma implementação eficiente nos níveis inferiores



## Evolução do Conjunto de Instruções





## Evolução do Conjunto de Instruções

- Maiores avanços em arquitetura de computadores são associados à mudanças no projeto do conjunto de instruções
  - Ex: Pilha vs GPR (Sistema 360)
- Decisões devem levar em conta:
  - tecnologia
  - organização de máquina
  - linguagens de programação
  - tecnologia de compiladores
  - sistemas operacionaisE eles influenciam...



## Métricas de um ISA

- Ortogonalidade
  - Nenhum registrador especial, poucas condições especiais, todos os modos de operando disponíveis com qualquer tipo de dado ou tipo de instrução
- Completude (*Completeness*)
  - Suporta gama variada de operações e aplicações
- Regularidade
  - Não devem haver *overloadings* para os diferentes campos da instrução
- Fácil entendimento
  - Recursos necessários devem ser facilmente determinados

Facilidade de compilação (programação?)

Facilidade de implementação

Escalabilidade



## Classifying ISAs

### Accumulator (before 1960):

1 address      add A       $\text{acc} \leftarrow \text{acc} + \text{mem}[A]$

### Stack (1960s to 1970s):

0 address      add       $\text{tos} \leftarrow \text{tos} + \text{next}$

### Memory-Memory (1970s to 1980s):

2 address      add A, B       $\text{mem}[A] \leftarrow \text{mem}[A] + \text{mem}[B]$

3 address      add A, B, C       $\text{mem}[A] \leftarrow \text{mem}[B] + \text{mem}[C]$

### Register-Memory (1970s to present):

2 address      add R1, A       $R1 \leftarrow R1 + \text{mem}[A]$

load R1, A       $R1 \leftarrow \text{mem}[A]$

### Register-Register (Load/Store) (1960s to present):

3 address      add R1, R2, R3       $R1 \leftarrow R2 + R3$

load R1, R2       $R1 \leftarrow \text{mem}[R2]$

store R1, R2       $\text{mem}[R1] \leftarrow R2$



## Classes de ISAs

### 1. Acumulador:

1 endereço      add A       $\text{acc} \leftarrow \text{acc} + \text{mem}[A]$

### 2. Pilha:

0 endereços      add       $\text{tos} \leftarrow \text{tos} + \text{próximo}$

### 3. General Purpose Register (GPR):

2 endereços      add A B       $\text{EA}(A) \leftarrow \text{EA}(A) + \text{EA}(B)$

3 endereços      add A B C       $\text{EA}(A) \leftarrow \text{EA}(B) + \text{EA}(C)$

0 endereços (Load/Store)

add Ra Rb Rc       $Ra \leftarrow Rb + Rc$

load Ra Rb       $Ra \leftarrow \text{mem}[Rb]$

store Ra Rb       $\text{mem}[Rb] \leftarrow Ra$



## Acumulador

- Operando é o registrador acumulador

$A = B + C$

ld B  
add C  
sto A

- Atributos
  - Instruções compactas
  - Estado interno mínimo, projeto interno simples
  - Alto tráfego de memória, muitos loads e stores
- Exemplos:
  - IBM 7090, DEC PDP-8, MOS 6502, 8085



## Máquinas de Pilha

- Conjunto de instrução:

+, -, \*, /, ...

push A, pop A

- Exemplo:  $a*b - (a+c*b)$

push a

push b

\*

push a

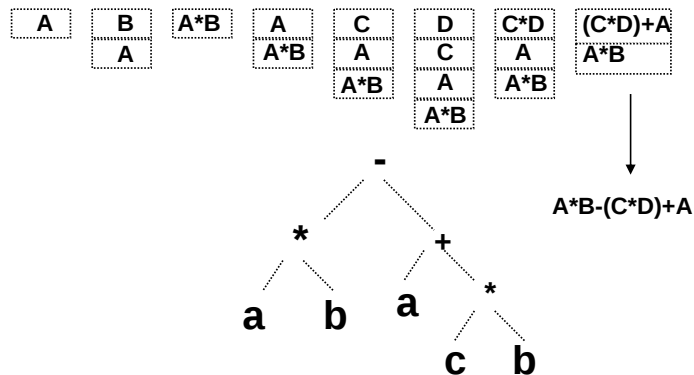
push c

push b

\*

+

-





## Máquinas de Pilha

- Não utilizam registradores
- Não possuem operandos específicos para a ALU
- Vantagens:
  - Instruções compactas, endereço dos operandos implícito na pilha
  - Compilador é fácil de escrever
- Exemplos: Burroughs B5500/6500, HP 3000/70, 8087 (co-processador de ponto flutuante do 486 e Pentium)



## Máquinas de Pilha: Desvantagens

- Performance é devido à existência de diversos registradores rápidos e não à maneira como são organizados
- Dados não aparecem no topo da pilha quando precisamos deles
  - Constantes, operandos repetidos
- Densidade de código é igual à densidade de máquinas com GPR
  - Registradores possuem campo de endereço com alta densidade
  - Mantenha resultados intermediários nos registradores para reutilizá-los
- Compilador é fácil de escrever, difícil de otimizar



## Máquinas com Registradores de Uso Geral (GPR)

- Registradores considerados memória rápida bem próxima a ALU
- Opções:
  - 2 vs. 3 operandos:
    - 2 operandos  $R = R \text{ op } y$
    - 3 operandos  $R = x \text{ op } y$
  - Operações de ALU podem ou não acessar memória
    - RISC (L/S ou R/R) : não
    - CISC (R/M) : sim

$A = B + C$

ld r1, B  
add r1, C  
st A, r1

ld r1, B  
ld r2, C  
add r3, r1, r2  
st A, r3



## Máquinas com Registradores de Uso Geral (GPR)

- Arquitetura dominante atualmente: CDC 6600, IBM 360/370, PDP-11, 68K, 386+, RISCs, etc...
- Vantagens:
  - Permite acesso mais rápido a valores temporários
  - Permite técnicas mais agressivas de compilação
  - Reduz tráfego à memória
- Desvantagens:
  - Instruções mais longas
  - Troca de contexto mais lenta



## Máquinas GPR

- L/S ou R/R (0,3)
  - Tipicamente 3 operandos
  - Exemplos: CDC 6600, Cray 1, RISCs
  - Vantagens:
    - Codificação simples das instruções
    - Fácil de gerar código e *pipeline*
    - Ciclos/instrução são +/- iguais
  - Desvantagens:
    - IC mais alto
    - Desperdício de códigos



## Máquinas GPR

- R/M (1,2)
  - 1 operando de memória, 2 operandos
  - Exemplos: 80x86
  - Vantagens:
    - Acesso a dados sem carregar registradores
    - Boa densidade de código
  - Desvantagens:
    - Operandos não são equivalentes
    - Codificação de operandos limita número de registradores
    - Variância de CPI é mais alta
    - Mais difícil de *pipeline*



## Máquinas GPR

- R/M (3,3)
  - 3 operandos de memória, 3 operandos
  - Exemplos: VAX
  - Vantagens:
    - Compactação e não utiliza registradores como temporários
  - Desvantagens:
    - Variação alta no tamanho da instrução
    - Gargalo na memória (1 instr, 4 referências)
    - Variância de CPI é a mais alta de todas
    - Muito mais difícil de *pipeline*



## Instruction Set Design Issues

- Instruction set design issues include:
  - Where are operands stored?
    - registers, memory, stack, accumulator
  - How many explicit operands are there?
    - 0, 1, 2, or 3
  - How is the operand location specified?
    - register, immediate, indirect, . . .
  - What type & size of operands are supported?
    - byte, int, float, double, string, vector. . .
  - What operations are supported?
    - add, sub, mul, move, compare . . .



# Espaço de Projeto de uma ISA

## Dimensões primárias

- Número de operandos explícitos ( 0, 1, 2, 3 )
- Armazenamento de operandos Onde além da memória?
- Endereço efetivo Como endereço de memória é especificado?
- Tipo e tamanho dos operandos byte, int, float, vector, . . .
- Operações Como especificá-lo?  
add, sub, mul, . . .  
Como especificá-lo?

## Outros aspectos

- Sucessor Como é especificado?
- Condições Como são determinadas?
- Codificações Fixa ou variável? *Wide*?
- Paralelismo

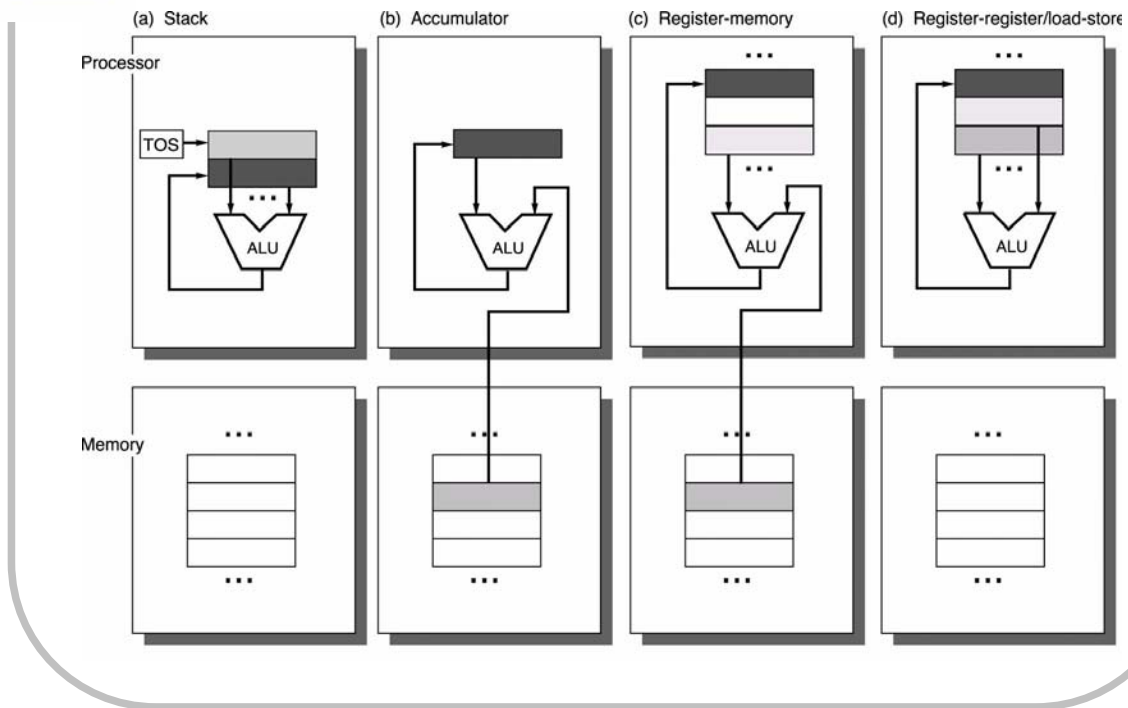


# Operands

- How are operands designated?
  - fixed – always in the same place
  - by opcode – always the same for groups of instructions
  - by a field in the instruction – requires decode first
- What is the format of the data?
  - binary
  - character
  - decimal (packed and unpacked)
  - floating-point – IEEE 754 (others used less and less)
  - size – 8-, 16-, 32-, 64-, 128-bit
- What is the influence on ISA?



## Operand Locations



## Operations

- Types
  - ALU – Integer arithmetic and logical functions
  - Data transfer – Loads/stores
  - Control – Branch, jump, call, return, traps, interrupts
  - System – O/S calls, virtual memory management
  - Floating point – Floating point arithmetic
  - Decimal – Decimal arithmetic
  - String – moves, compares, search, etc.
  - Graphics – Pixel/vertex operations
  - Vector – Vector (SIMD) functions
- Addressing
  - Which addressing modes for which operands are supported?



# Operations

- Types
  - ALU – Integer arithmetic and logical functions
  - Data transfer – Loads/stores
  - Control – Branch, jump, call, return, traps, interrupts
  - System – O/S calls, virtual memory management
  - Floating point – Floating point arithmetic
  - Decimal – Decimal arithmetic
  - String – moves, compares, search, etc.
  - Graphics – Pixel/vertex operations
  - Vector – Vector (SIMD) functions
- Addressing
  - Which addressing modes for which operands are supported?



# Addressing Modes

- Types
  - Register – data in a register
  - Immediate – data in the instruction
  - Memory – data in memory
- Calculation of *Effective Address*
  - Direct – address in instruction
  - Indirect – address in register
  - Displacement – address = register or PC + offset
  - Indexed – address = register + register
  - Memory Indirect – address at address in register
- What is the influence on ISA?



## Types of Addressing Mode (VAX)

| Addressing Mode      | Example             | Action                                               |
|----------------------|---------------------|------------------------------------------------------|
| 1. Register direct   | Add R4, R3          | $R4 \leftarrow R4 + R3$                              |
| 2. Immediate         | Add R4, #3          | $R4 \leftarrow R4 + 3$                               |
| 3. Displacement      | Add R4, 100(R1)     | $R4 \leftarrow R4 + M[100 + R1]$                     |
| 4. Register indirect | Add R4, (R1)        | $R4 \leftarrow R4 + M[R1]$                           |
| 5. Indexed           | Add R4, (R1 + R2)   | $R4 \leftarrow R4 + M[R1 + R2]$                      |
| 6. Direct            | Add R4, (1000)      | $R4 \leftarrow R4 + M[1000]$                         |
| 7. Memory Indirect   | Add R4, @(R3)       | $R4 \leftarrow R4 + M[M[R3]]$                        |
| 8. Autoincrement     | Add R4, (R2)+       | $R4 \leftarrow R4 + M[R2]$<br>$R2 \leftarrow R2 + d$ |
| 9. Autodecrement     | Add R4, (R2)-       | $R4 \leftarrow R4 + M[R2]$<br>$R2 \leftarrow R2 - d$ |
| 10. Scaled           | Add R4, 100(R2)[R3] | $R4 \leftarrow R4 + M[100 + R2 + R3*d]$              |

- Studies by [Clark and Emer] indicate that *modes 1-4 account for 93% of all operands on the VAX*



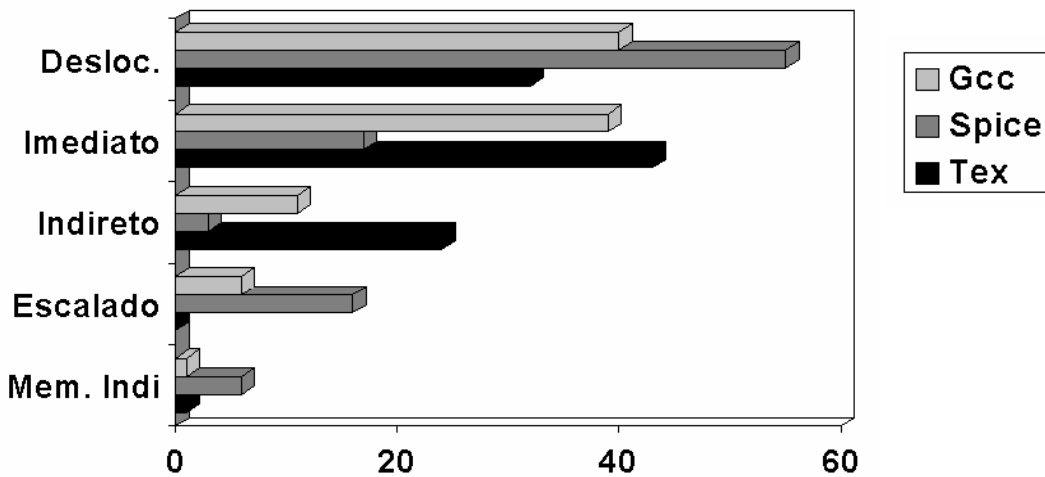
## Modos de Endereçamento

| Modo de End.        | Exemplo            | Significado                                    | Utilização            |
|---------------------|--------------------|------------------------------------------------|-----------------------|
| Registrador         | ADD R4,R3          | $Regs[R4] += Regs[R3]$                         | Valores em regs       |
| Imediato            | ADD R4,#2          | $Regs[R4] += 2$                                | Constantes            |
| Relativo            | ADD R4,100(R1)     | $Regs[R4] += Mem[Regs[R1] + 100]$              | Vars. locais          |
| Indexado            | ADD R4,(R1 + R2)   | $Regs[R4] += Mem[Regs[R1] + Regs[R2]]$         | Vetores               |
| Absoluto ou Direto  | ADD R4,(1001)      | $Regs[R4] += Mem[1001]$                        | Dados estáticos       |
| Indireto            | ADD R4,(R1)        | $Regs[R4] += Mem[Regs[R1]]$                    | Pointers              |
| Indireto em memória | ADD R4,@(R1)       | $Regs[R4] += Mem[Mem[Regs[R1]]]$               | Pointer duplo ou *p   |
| Autoincremento      | ADD R4,(R1)+       | $Regs[R4] += Mem[Regs[R1]]$<br>$Regs[R1] += d$ | Pilha (push) ou array |
| Autodecremento      | ADD R4,-(R1)       | $Regs[R1] -= d$<br>$Regs[R4] += Mem[Regs[R1]]$ | Pilha (pop) ou array  |
| Escalado            | ADD R4,100(R1)[R2] | $Regs[R4] += Mem[Regs[R1] + Regs[R2] + 100]$   | Arrays locais         |

- Reduzem IC
- Aumentam CPI



## Modos de Endereçamento



- Somente modos com frequência acima de 1%
- Deslocamentos de 8, 16 e 32 bits
- Modo registrador não foi considerado (50% dos operandos)



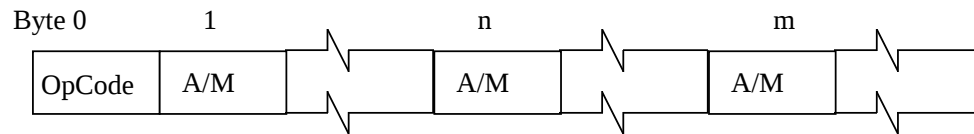
## Instruction Encoding

- Variable
  - Instruction length varies based on opcode and address specifiers
  - For example, VAX instructions vary between 1 and 53 bytes, while x86 instructions vary between 1 and 17 bytes.
  - Good code density, but difficult to decode and pipeline
- Fixed
  - Only a single size for all instructions
  - For example MIPS, Power PC, Sparc all have 32 bit instructions
  - Not as good code density, but easier to decode and pipeline
- Hybrid
  - Have multiple format lengths specified by the opcode
  - For example, IBM 360/370
  - Compromise between code density and ease of decode



# VAX-11

## Formato variável, instruções com 2 ou 3 endereços



**Palavra de 32-bits, 16 GPRs (4 reservados)**

- Conjunto rico de endereçamentos (aplicáveis a qualquer operando)
- Conjunto rico de operações
  - bit field, stack, call, case, loop, string, poly, system
- Conjunto rico de tipos (B, W, L, Q, O, F, D, G, H)
- *Condition codes*

**O que realmente foi utilizado?**