Aula 3

Engenharia de Sistemas Embarcados

Prof. Abel Guilhermino

Tópico: Projeto de Sistemas Embarcados

Arquiteturas VLIW

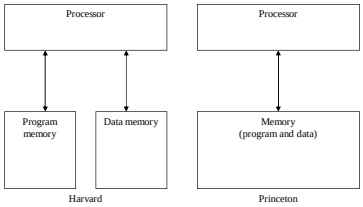
- As arquiteturas VLIW (Lery Long Instruction Word).
- Tenta alcançar maiores níveis de paralelismo de instrução pela execução de instruções longas compostas por múltiplas operações.
- Consistem de várias operações artiméticas, lógicas e de controle cada uma das quais poderia ser uma operação individual em um processador RISC simples.
- O processador VLIW executa o conjunto de operações concorrentemente, alcançando assim um alto grau de paralelismo no nível de instrução.
- É responsabilidade do compilador escalonar as operações de modo a utilizar o melhor possível as unidades funcionais disponíveis no processador.
  - O compilador descobre as instruções que podem ser executadas em paralelo e agrupa-as formando uma longa instrução.
- Um dos maiores obstáculos à evolução das arquiteturas VLIW tem sido a falta de compatibilidade binária com as arquiteturas convencionais.

Arquiteturas Superescalares x VLIW

Superescalar	VLIW
Detecção de paralelismo	
hardware	compilador
Tempo disponível para realizar a detecção	
pouco	muito
Relógio	
mais lento	mais rápido
Arquitetura	
CISC	RISC

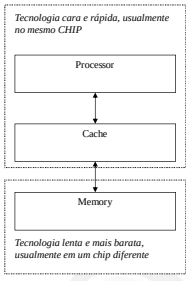
Arquiteturas com Duas Memórias

- Princeton
  - Poucas conexões de memórias
- Harvard
  - Acesso simultâneo a dados e programas



Memória Cache

- Acesso a memória pode ser lento
- Cache é uma memória pequena mas é rápida e próxima ao processador
  - Contém cópia de parte da memória
  - Hits e misses



Visão do Programador

- Programador não precisa de entendimento detalhado da arquitetura
  - Ao invés disso, precisa saber que instruções podem ser executadas
- Dois níveis de instrução
  - Nível assembly
  - Linguagens estruturadas (C, C++, Java, etc.)
- Grande parte do desenvolvimento é feito usando linguagens estruturadas
  - Mas, algum nível de programação assembly ainda pode ser necessário
  - Drivers: parte do programa que se comunica com e/ou controla outro dispositivo
    - Com freqüência contém considerações detalhadas de temporização, manipulação em larga escala de bits
    - Nível assembly pode ser melhor neste caso

Projeto das Instruções

Y = (A – B) / (C + D x E)

Instrução	Comentário
LOAD D	AC ← D
MUL E	AC ← AC x E
ADD C	AC ← AC + C
STOR Y	Y ← AC
LOAD A	AC ← A
SUB B	AC ← AC – B
DIV Y	AC ← AC / Y
STOR Y	Y ← AC

Instruções com 3 endereços

Instruções com 2 endereços

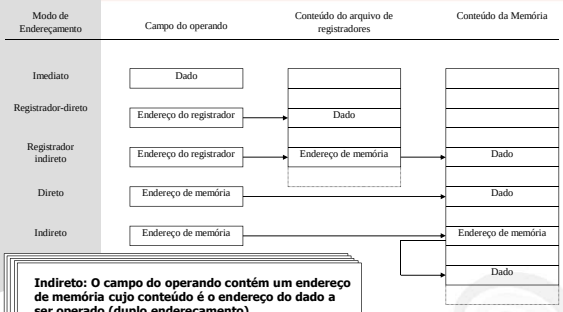
Instruções com 1 endereços

- Menor número de endereços → instruções de menor extensão
- Menor número de endereços → instruções mais primitivas → CPU menos complexa
- Menor número de endereços → Mais instruções no programa → maior tempo de execução & programas mais complexos
- Apenas um endereço → Geralmente só existe um registrador disponível para o programador
- Maior número de endereços → Mais registradores disponíveis para propósito geral → operações efetuadas sobre registradores → execução mais rápida
- Maior número de endereços → Maior flexibilidade → maior aplicação a maquinas modernas

Um Conjunto Simples de Instruções

Assembly instruct.	Primeiro byte		segundo byte		Operação
MOV Rn, direto	0000	Rn	direto		Rn = M(direto)
MOV direto, Rn	0001	Rn	direto		M(direto) = Rn
MOV @Rn, Rm	0010	Rn	Rm		M(Rn) = Rm
MOV Rn, #immed.	0011	Rn	imediato		Rn = imediato
ADD Rn, Rm	0100	Rn	Rm		Rn = Rn + Rm
SUB Rn, Rm	0101	Rn	Rm		Rn = Rn - Rm
JZ Rn, relativo	0110	Rn	relativo		PC = PC+ relativo (somente se Rn é 0)
	opcode		operandos		

Modos de Endereçamento



Indireto: O campo do operando contém um endereço de memória cujo conteúdo é o endereço do dado a ser operado (duplo endereçamento)
-Vant: Elimina a limitação de células endereçaveis
-Desv: Mais acessos à MP

Programas Exemplos

Programa em C

```
int total = 0;
for (int i=10; i!=0; i--)
  total += i;
// next instructions...
```

Programa equivalente em assembly

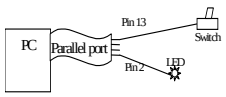
```
0     MOV R0, #0;      // total = 0
1     MOV R1, #10;     // i = 10
2     MOV R2, #1;      // constant 1
3     MOV R3, #0;      // constant 0
Loop: JZ R1, Next;     // Done if i=0
5     ADD R0, R1;      // total += i
6     SUB R1, R2;      // i--
7     JZ R3, Loop;     // Jump always
Next: // next instructions...
```


Considerações do Programador

- Espaço de memória de dados e programa
  - Com freqüência é limitado em processadores embarcados
  - Ex: 64 Kbytes p/ programa, 256 bytes dados
- Registradores: quantos são?
  - Preocupação direta para programadores assembly
- E/S
  - Como se comunicar com sinais externos?
- Interrupções

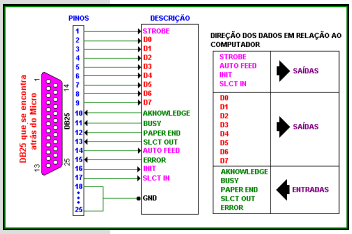
Exemplo: Driver da Porta Paralela

LPT Pino de conexão	Direção de E/S	Endereço do Registrador
1	saída	0th bit of register #2
2-9	saída	0th bit of register #2
10,11,12,13,15	entrada	6,7,5,4,3th bit of register #1
14,16,17	saída	1,2,3th bit of register #2

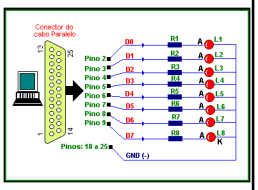


- Utilizando programação em linguagem assembly podemos configurar uma porta paralela do PC para realizar E/S digital
  - Exemplo: a porta paralela pode ser utilizada para monitorar a entrada da chave e ligar ou desligar o LED de acordo com a entrada

Exemplo de Programa para a Porta Paralela



Circuito



Componentes

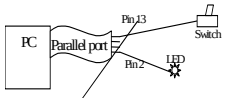
8 resistores (R1 a R8) 470 Ohms
8 leds (L1 a L8) comuns

Nome	Endereços LPT1	Endereços LPT2	Descrição
Registro de Dados	378h	278h	Envia um byte para a impressora
Registro de Status	379h	279h	Ler o Status da impressora
Registro de Controle	37Ah	27Ah	Envia dados de controle para a impressora

Exemplo de Programa para a Porta Paralela

```
; This program consists of a sub-routine that reads
; the state of the input pin, determining the on/off state
; of our switch and asserts the output pin, turning the LED
; on/off accordingly
        .386

CheckPort   proc
        push    ax              ; save the content
        push    dx              ; save the content
        mov     dx, 3BCh + 1    ; base + 1 for register #1
        in      al, dx          ; read register #1
        and     al, 10h         ; mask out all but bit # 4
        cmp     al, 0           ; is it 0?
        jne     SwitchOn        ; if not, we need to turn the LED on

SwitchOff:
        mov     dx, 3BCh + 0 ; base + 0 for register #0
        in      al, dx          ; read the current state of the port
        and     al, f7h         ; clear first bit (masking)
        out     dx, al          ; write it out to the port
        jmp     Done            ; we are done

SwitchOn:
        mov     dx, 3BCh + 0 ; base + 0 for register #0
        in      al, dx          ; read the current state of the port
        or      al, 01h         ; set first bit (masking)
        out     dx, al          ; write it out to the port

Done:   pop     dx              ; restore the content
        pop     ax              ; restore the content
CheckPort   endp
```

```
extern "C" CheckPort(void);         // defined in
                                    // assembly
void main(void) {
        while( 1 ) {
                CheckPort();
        }
}
```



LPT Pino de conexão	Direção de E/S	Endereço de Registrador
1	saída	0th bit of register #2
2-9	Saída	0th bit of register #2
10,11,12,13,15	entrada	6,7,5,4,3th bit of register #1
14,16,17	saída	1,2,3th bit of register #2

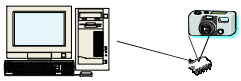
Sistema Operacional

- Camada de software opcional que provê serviços de baixo nível para o programa (aplicação)
  - Gerenciamento de arquivos, acesso a disco
  - Interface com teclado e display
  - Escalonamento de múltiplos programas para execução
    - Ou até múltiplo threads em um programa
  - Programa realiza chamadas de sistema para o SO

```
DB file_name "out.txt" -- store file name

MOV R0, 1324          -- system call "open" id
MOV R1, file_name     -- address of file-name
INT 34                -- cause a system call
JZ  R0, L1            -- if zero -> error

   . . . read the file
JMP L2                -- bypass error cond.
L1:
   . . . handle the error

L2:
```


Ambiente de Desenvolvimento

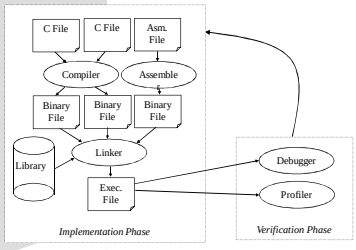
- Processador de desenvolvimento
  - Processador no qual escrevemos e depuramos nossos programas
    - Usualmente um PC
- *Processador alvo*
  - *Processador no qual o programa será executado em nosso sistema embarcado*
    - *Com freqüência diferente do processador de desenvolvimento*



Processador de desenvolvimento

Processador alvo

Processo de Desenvolvimento de Software



- Compiladores
  - Cross compiler
    - Roda em um processador mas gera código para outro
- Assemblers
- Linkers
- Debuggers
- Profilers

Executando um programa

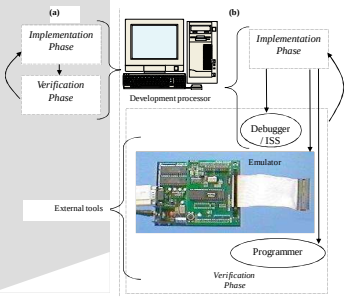
- Se o processador de desenvolvimento é diferente do processador alvo, como podemos executar o código compilado?
  - Carregar (download) para o processador alvo
  - simular
- Simulação
  - Método 1: Hardware description language
    - lento, nem sempre disponível
  - Método 2: *Instruction set simulator (ISS)*
    - Roda no processador de desenvolvimento, mas executa instruções do processador alvo.

Instruction Set Simulator de um processador simples

```
#include <stdio.h>
typedef struct {
   unsigned char first_byte, second_byte;
} instruction;

instruction program[1024];  //instruction memory
unsigned char memory[256];  //data memory

void run_program(int num_bytes) {

   int pc = -1;
   unsigned char reg[16], fb, sb;

   while( ++pc < (num_bytes / 2) ) {
      fb = program[pc].first_byte;
      sb = program[pc].second_byte;
      switch( fb >> 4 ) {
         case 0: reg[fb & 0x0f] = memory[sb]; break;
         case 1: memory[sb] = reg[fb & 0x0f]; break;
         case 2: memory[reg[fb & 0x0f]] =
                    reg[sb >> 4];  break;
         case 3: reg[fb & 0x0f] = sb; break;
         case 4: reg[fb & 0x0f] += reg[sb >> 4]; break;
         case 5: reg[fb & 0x0f] -= reg[sb >> 4]; break;
         case 6: pc += sb; break;
         default: return -1;
```

```
      }
   }
   return 0;
}

int main(int argc, char *argv[]) {

   FILE* ifs;

   If( argc != 2 ||
       (ifs = fopen(argv[1], "rb") == NULL ) {
             return -1;
   }
   if (run_program(fread(program,
       sizeof(program) == 0) {
             print_memory_contents();
             return(0);
   }
   else return(-1);
}
```


Testando e Depurando



- ISS
  - Gives us control over time - set breakpoints, look at register values, set values, step-by-step execution, ...
  - But, doesn't interact with real environment
- Download to board
  - Use device programmer
  - Runs in real environment, but not controllable
- Compromise: emulator
  - Runs in real environment, at speed or near
  - Supports some controllability from the PC

Application-Specific Instruction-Set Processors (ASIPs)

- General-purpose processors
  - Sometimes too general to be effective in demanding application
    - e.g., video processing - requires huge video buffers and operations on large arrays of data, inefficient on a GPP
  - But single-purpose processor has high NRE, not programmable
- ASIPs - targeted to a particular domain
  - Contain architectural features specific to that domain
    - e.g., embedded control, digital signal processing, video processing, network processing, telecommunications, etc.
  - Still programmable

A Common ASIP: Microcontroller

- For embedded control applications
  - Reading sensors, setting actuators
  - Mostly dealing with events (bits): data is present, but not in huge amounts
  - e.g., VCR, disk drive, digital camera (assuming SPP for image compression), washing machine, microwave oven
- Microcontroller features
  - On-chip peripherals
    - Timers, analog-digital converters, serial communication, etc.
    - Tightly integrated for programmer, typically part of register space
  - On-chip program and data memory
  - Direct programmer access to many of the chip's pins
  - Specialized instructions for bit-manipulation and other low-level operations

Another Common ASIP: Digital Signal Processors (DSP)

- For signal processing applications
  - Large amounts of digitized data, often streaming
  - Data transformations must be applied fast
  - e.g., cell-phone voice filter, digital TV, music synthesizer
- DSP features
  - Several instruction execution units
  - Multiple-accumulate single-cycle instruction, other instrs.
  - Efficient vector operations - e.g., add two arrays
    - Vector ALUs, loop buffers, etc.

Trend: Even More Customized ASIPs

- In the past, microprocessors were acquired as chips
- Today, we increasingly acquire a processor as Intellectual Property (IP)
  - e.g., synthesizable VHDL model
- Opportunity to add a custom datapath hardware and a few custom instructions, or delete a few instructions
  - Can have significant performance, power and size impacts
  - Problem: need compiler/debugger for customized ASIP
    - Remember, most development uses structured languages
    - One solution: automatic compiler/debugger generation
      - e.g., www.tensillica.com
    - Another solution: retargettable compilers
      - e.g., www.improvsys.com (customized VLIW architectures)

Selecting a Microprocessor

- Issues
  - Technical: speed, power, size, cost
  - Other: development environment, prior expertise, licensing, etc.
- Speed: how evaluate a processor's speed?
  - Clock speed - but instructions per cycle may differ
  - Instructions per second - but work per instr. may differ
  - Dhrystone: Synthetic benchmark, developed in 1984. Dhrystones/sec.
    - MIPS: 1 MIPS = 1757 Dhrystones per second (based on Digital's VAX 11/780). A.k.a. Dhrystone MIPS. Commonly used today.
      - So, 750 MIPS = 750*1757 = 1,317,750 Dhrystones per second
  - SPEC: set of more realistic benchmarks, but oriented to desktops
  - EEMBC - EDN Embedded Benchmark Consortium, www.eembc.org
    - Suites of benchmarks: automotive, consumer electronics, networking, office automation, telecommunications

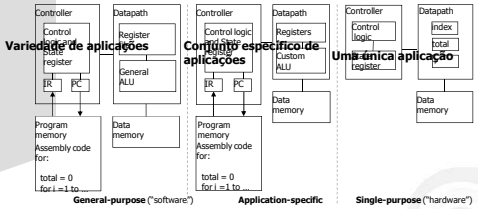
General Purpose Processors

Processor	Clock speed	Periph.	Bus Width	MIPS	Power	Trans.	Price
General Purpose Processors							
Intel PIII	1GHz	2x16 K L1, 256K L2, MMX	32	~900	97W	~7M	$900
IBM PowerPC 750X	550 MHz	2x32 K L1, 256K L2	32/64	~1300	5W	~7M	$900
MIPS R5000	250 MHz	2x32 K 2 way set assoc.	32/64	NA	NA	3.6M	NA
StrongARM SA-110	233 MHz	None	32	268	1W	2.1M	NA
Microcontroller							
Intel 8051	12 MHz	4K ROM, 128 RAM, 32 I/O, Timer, UART	8	~1	~0.2W	~10K	$7
Motorola 68HC811	3 MHz	4K ROM, 192 RAM, 32 I/O, Timer, WDT, SPI	8	~.5	~0.1W	~10K	$5
Digital Signal Processors							
TI C5416	160 MHz	128K, SRAM, 3 T1 Ports, DMA, 13 ADC, 9 DAC	16/32	~600	NA	NA	$34
Lucent DSP32C	80 MHz	16K Inst., 2K Data, Serial Ports, DMA	32	40	NA	NA	$75

Sources: Intel, Motorola, MIPS, ARM, TI, and IBM Website/Datasheet; Embedded Systems Programming, Nov. 1998

Tecnologia dos Processadores

- Arquitetura do componente de computação que implementa a funcionalidade desejada.
- Vários tipos de processadores podem implementar esta funcionalidade:



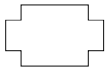
Tecnologia de Processadores

- Processadores podem variar na adequação ao problema

Funcionalidade Desejada

```
total = 0
for i = 1 to N  loop
   total += M[i]
end loop
```

Processador de Propósito geral



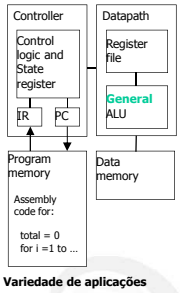
ASIP – Processador de Aplicação Específica

Processador de propósito único

(Application-Specific Instruction-set Processor)

Processador de uso geral

- Programados via software
- Memória para dados e programa
- Vantagens
  - Pequenos time-to-market e custo não recorrente (NRE)
  - Alta flexibilidade
- Desvantagens
  - Necessidade de Adicionar dispositivos
  - Alto consumo
  - Baixo desempenho
- Ex: Power PC, Pentium, Z80



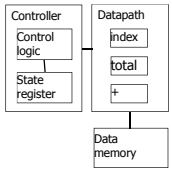
Variedade de aplicações

Processadores Embarcados (embedded)

- Processadores de uso geral adaptados para sistemas embarcados
  - Dispositivos Internos
  - Menor potência
  - Facilidade para desenvolver software

Processador de Propósito Único

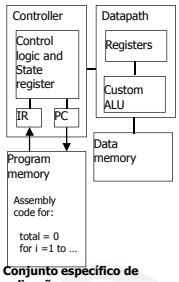
- Circuitos Digital projetado para executar exatamente um único programa
  - Ex: um co-processador, acelerador ou periférico
- Características
  - Contém apenas componentes necessários para executar um único programa.
  - Nenhuma memória de programa
- Vantagens
  - Rápido
  - Baixa Potência
  - Pequeno



Uma única aplicação

Processadores de Aplicação Específica

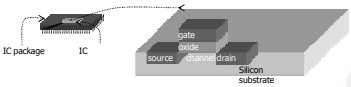
- Processsdores programáveis otimizados para uma classe particular de aplicações (ASIPs)
- Características
  - Compromisso entre processadores de propósito geral e de propósito único.
  - Memória de programa
  - Datapath otimizado
  - Unidades funcionais especiais
- Vantagens
  - Alguma flexibilidade, bom desempenho, tamanho e potência



Conjunto específico de aplicações

Tecnologia IC

- Maneira de mapear uma implementação digital (gate-level) em um circuito integrado
  - IC: Integrated circuit, or "chip"
  - Tecnologias de IC diferem no nível de especialização do projeto
  - CIs possuem inúmeros níveis (10 ou mais)
    - Tecnologias de IC diferem com respeito a quem constrói os níveis



IC technology

- Tipos de tecnologias IC
  - Full-custom/VLSI
  - Semi-custom ASIC (gate array e standard cell)
  - PLD (Programmable Logic Device)

Full-custom/VLSI

- Todos os níveis são otimizados para uma implementação particular de um sistema embarcado
  - Posicionamento dos transistores
  - Dimensionamento dos transistores
  - Roteamento dos fios
- Vantagens
  - Excelente desempenho, pequeno tamanho, baixa potência
- Desvantagens
  - Custo NRE alto (e.g., $300k), time-to-market bastante longo

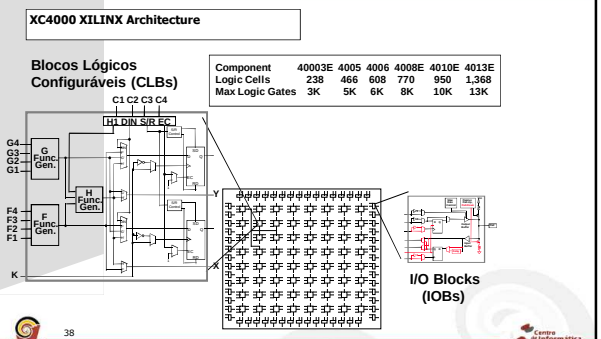
Semi-custom

- Níveis inferiores são completa ou parcialmente construídos
  - Projetistas devem rotear os fios ou posicionar alguns blocos
- Vantagens
  - Bom desempenho, bom tamanho, custo NRE menor que full-custom (de $10k a $100k)
- Desvantagens
  - Ainda requer semanas ou meses de desenvolvimento

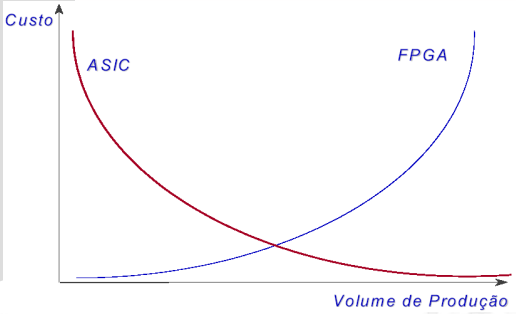
PLD (Programmable Logic Device)

- Todos os níveis já existem
  - Projetistas podem comprar como IC
  - Conexões são criadas ou destruídas para implementar funcionalidade desejada
  - Field-Programmable Gate Array (FPGA) são os mais populares
- Vantagens
  - Baixo custo NRE
- Desvantagens
  - Grandes, Caros ($30 por unidade), alto consumo de energia, lentos

FPGA: Estrutura Geral e Bloco Básico



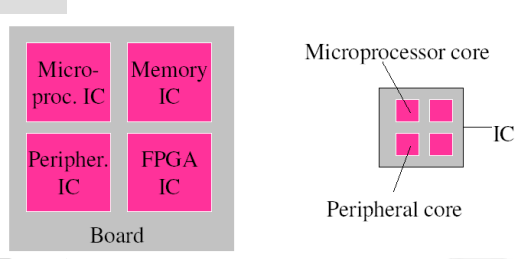
FPGA: Custo



System-On-a-Chip (SoC)

- Uso de núcleo de processadores (cores)
- Baixo Custo de fabricação em série
- Alta qualidade
- Diminuição de defeitos de montagem e fabricação em geral
- Baixa potência consumida
- Pequeno tamanho
- Alta velocidade

SoC



SoC