Universidade Federal do Rio Grande do Sul
Instituto de Informática

Organização de Computadores

Aula 2

Processadores RISC

Processadores RISC

1. **Motivação**

2. **Fundamentos RISC**

3. **Decisões de Projeto**

4. **Ganho na lógica de controle**

Introdução

- **Nos primeiros dias da indústria de computadores quase não havia tecnologia de compiladores,**
- **A programação era feita em código de máquina ou linguagem assembler,**
- **Para auxiliar a programação, os arquitetos de máquinas desenvolveram instruções cada vez mais complexas,**
- **Era mais fácil projetar em hardware do que fazer compiladores mais eficientes,**
- **Isto levou a um conjunto de instruções cada vez mais complexas**
- **CISC – Complex Instruction Set Computers**

1. Motivação

- **década de 60**
  - **introdução de famílias de computadores (ex: IBM 360)**
  - **distinção entre arquitetura e organização**
  - **uso de microprogramação**

- **arquitetura x organização**
  - **possível compromisso entre custo e desempenho não é definido unicamente pela arquitetura**
  - **impacto de uma instrução não é decisivo na implementação**

- **microprogramação**
  - **memória principal: núcleos, grande tempo de acesso**
  - **memória de controle: semicondutora, barata**
  - **grandes microprogramas não acrescentavam custo**
  - **maior desempenho: mover software para microprogramas**

Processadores CISC

- **Instruções mais complexas**
  - **facilitar tarefa dos compiladores**
  - **aumentar desempenho**

- **Digital VAX 11/780**
  - **303 instruções**
  - **16 modos de endereçamento**

- **Intel 386**
  - **111 instruções**
  - **8 modos de endereçamento**

- **Motorola 68020**
  - **109 instruções**
  - **18 modos de endereçamento**

- **Instruções com comprimento e formato variáveis**

Mudanças no contexto

- **A partir da metade da década de 70**
- **Memória principal: semicondutores no lugar de núcleos**
  - **memória principal não era mais 10 vezes mais lenta que memória de controle**
- **Introdução de memórias cache baratas e rápidas**
  - **acessos à memória cache tão rápidos quanto à memória de controle**
- **Efeitos colaterais de conjuntos complexos de instruções**
  - **dificuldade no desenvolvimento de microcódigos com até 400 Kbytes**
  - **tempo de projeto mais longo, mais erros de projeto**

Mudanças no contexto

- **Compiladores utilizando sub-conjuntos da arquitetura**
  - **difícil utilização de instruções complexas**
  - **otimização de código tornava possível utilização de instruções mais simples**
- **Maioria das instruções do conjunto eram pouco utilizadas nas aplicações mais comuns**
- **Advento de circuitos VLSI e problemas no projeto de processadores em um chip único**

Exemplo de uso de instruções

- **Exemplo: uso médio de instruções do 8086 em 3 aplicações – assembler MASM, compilador Turbo C, Lotus 1-2-3**

Transferência de dados	
MOV	29
PUSH / POP	12
LEA	3

Aritméticas / lógicas	
CMP	7
SAL / SHR / RCR	5
INC / DEC	5
ADD	3
OR / XOR	3

Controle / desvio	
JMP	2
LOOP	4
CALL / RET	4
desvios condicionais	10

2. Fundamentos RISC

- **Idéia lançada no final da década de 70 por pesquisadores da Universidade de Berkeley**
- **Desenvolver um conjunto de instruções pequeno e bem simples**
- **Máquina com arquitetura bem simples para poder ter organização mais eficiente e maior velocidade de operação**
- **Perspectiva global de aumento de desempenho**
- **Processador num chip único**
  - **melhor aproveitamento de recursos escassos**
  - **ganho na lógica de controle aproveitado no bloco operacional**

Fundamentos RISC

- **Maior número de instruções nos programas compensado por**
  - **instruções mais rápidas**
  - **instruções mais curtas**
- **Transferir para o software (compilador) o esforço de otimização do tempo de execução**
- **Dar suporte a linguagens de alto nível**
  - **escolher instruções que otimizem desempenho tendo em vista construções mais comuns em linguagens de alto nível**

CISC x RISC

- **CISC**
- **Ênfase no hardware**
- **Instruções com vários ciclos de relógio**
- **Instruções memória a memória**
- **Instruções complexas**
- **Load e Store estão incorporados nas instruções**
- **Tamanho de código pequeno, mas muitos ciclos por seg.**

- **RISC**
- **Ênfase no software**
- **Instruções de um ciclo de relógio**
- **Instruções registrador a registrador**
- **Instruções reduzidas**
- **Load e store são instruções independentes**
- **Tamanha de código largo mas poucos ciclos por seg.**

Vantagens da Arquitetura RISC (1)

- **Universalmente aceito que Risc :**
  - **Minimiza o tempo de execução**
  - **Minimiza os custos de projeto**
- **Maneiras de alcançar estas metas:**
  - **Incrementar a tecnologia dos componentes**
  - **Minimizar o número médio de ciclos de clock por instrução**
  - **Execução simultânea de diversas instruções**
- **Arquiteturas CISC precisam de mais lógica complexa**
  - **Um microprograma complexo é necessário**
  - **Uma parte substancial da área do chip é ocupado pelo microcódigo**
- **Tamanho da área do chip necessária para a unidade de controle:**

  CISC → 40%..60% RISC → 10%
- **A área que sobra numa arquitetura RISC pode ser empregada por outros componentes.**

Vantagens da Arquitetura RISC (2)

- **Velocidade de processamento**
  - **Arquiteturas RISC são mais adaptadas para usarem instruções pipelines**
- **Implementação VLSI**
  - **A unidade de controle da arquitetura RISC é em hardware (não microprogramada)**
    - **Um grande número de registradores e memórias cache on-chip reduzem o número de acessos à memória**
- **Tempo de projeto**
  - **O tempo necessário para testar e depurar o harware é menor**
  - **As possibilidades de erros de projetos são menores**
  - **Vantagens: menores custos de projeto; maior confiabilidade no projeto**
- **Suporte à linguagem de Alto-Nível**
  - **Instruções semanticamente próximas das características das linguagens de alto-nível**

3. Decisões de Projeto

- **Fase de execução da instrução num <u>único ciclo de relógio</u>**
  - **instruções tão rápidas quanto micro-instruções**

- **Todas as <u>instruções do mesmo tamanho</u> e com o mesmo formato (ou com poucas variações de formato)**
  - **simplificar implementação do controle**

- **<u>Dados imediatos</u> pequenos e <u>deslocamentos pequenos</u>**

- **Acesso à memória principal apenas através de <u>instruções LOAD e STORE</u>**
  - **demais instruções fazem operações apenas entre registradores**
  - **simplificar implementação do controle (*pipeline*)**
  - **tornar operações aritméticas e lógicas mais rápidas**

Decisões de Projeto

- **Usar modos de endereçamento bem simples**
  - **simplificar implementação do controle**
- **Poucos tipos de dados**
- **Usar arquitetura Harvard**
  - **memórias de dados e instruções separadas permitem aumentar *bandwidth* de memória**
- **Uso de instruções *compare-and-branch***
  - **estatísticas mostram que até 98% dos branches são precedidos por uma comparação**
  - **comparação e branch reunidos numa única instrução evitam necessidade de flags (N, Z) e permitem execução num único ciclo**

Decisões dos Projetistas

- **Aumentar o tamanho do conjunto de registradores,**
- **Aumentar paralelismo interno,**
- **Aumentar o tamanho das caches,**
- **Acrescentar funcionalidades, E/S, timers,**
- **Adicionar processadores vetoriais**
- **Produzir os chips em fábricas de tecnologias mais antigas**

Característica geralmente encontradas no RISC

- **Código de instruções uniforme, Exemplo o código de operação é sempre encontrado na mesma posição das instruções,**
- **Registradores homogêneos. Podem ser empregados em qualquer contexto e facilitam o funcionamento dos compiladores,**
- **Modos de endereçamento simplificados, As instruções complexas com endereçamentos complexos foram substituídas por uma seqüência de instruções simples,**
- **Poucos tipos de dados suportados. Não existe, como no CISC suporte a instruções que trabalham com campos de bytes de algum dado.**

Primeiros RISC

- **David Patterson inicia em 1980 o projeto UC Berkeley´s RISC project, para ganhar desempenho através do uso de pipeline e um uso mais agressivo de registradores**
- **O projeto gerou o RISC-I em 1982, com 44.000 transistores e 32 instruções,**

- **John Henessy em Stanford inciou na mesma época 1981 o projeto do MIPS, baseado no uso intensivo do pipeline. Com todas as instruçõe sendo completadas em 1 ciclo de máquina**

- **A IBM inicia em 1975 o projeto de uma CPU que iria se chamar IBM 801. Este projeto lança em 1981 um chip incorporando idéias do RISC, era uma CPU para executar mini tarefas. O processador não foi um sucesso, mas foi a base para a criação da linha Power**

Exemplo

V1 = V2 + V3 + V4 + V5
V1 a V5 são variáveis em memória

Processador CISC

supondo instruções com 3 endereços, ocupando 3 palavras

```
V1 = V2 + V3
V1 = V1 + V4
V1 = V1 + V5
```

3 x 3 = 9 palavras
3 x 6 = 18 acessos à memória

Processador RISC

instruções ocupam 1 palavra

```
LOAD   R2, V2
LOAD   R3, V3
ADD    R1, R2, R3
LOAD   R4, V4
ADD    R1, R1, R4
LOAD   R5, V5
ADD    R1, R1, R5
STORE  V1, R1
```

8 x 1 = 8 palavras
8 x 1 + 5 x 1 = 13 acessos à memória

Resumo das Características

1. Uma arquitetura RISC é uma máquina load/store

3. A maioria das execuções de instruções são em 1 ciclo de clock

5. Instruções possuem um formato fixo

7. A unidade de controle é em hardware

9. Existe um pequeno número de formatos para instruções

11. A CPU possui um grande número de registros
    - Solução: uma grande memória cache on-chip

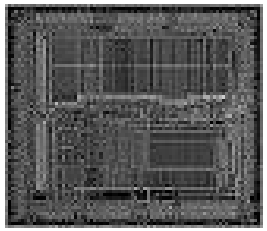
4. Ganho na lógica de controle

- **Uso de lógica *hardwired*, e não microprogramada – maior velocidade**
- **Microprocessadores convencionais ocupam até 50% do espaço com bloco de controle**
- **Processadores RISC: controle ocupa apenas 10% do espaço**
- **Aumento do número de registradores no espaço ganho**
- **Maior possibilidade de uso de *pipelines* homogêneos**

Alguns Chips RISC

- **SPARC da SUN Microsystems**

- **R2000 da MIPS Computer Systems, encontrado em máquinas SGI**



- **RS/6000 da IBM linha POWER**

- **POWER PC empregado pela Apple**
- **DEC Alpha**

- **HP com PA-RISC**

FIM