Projecto (Integrado) de xxx
Relatório da Fase xx
Grupo yyyy

Autor1 (número) Autor2 (número) Autor3 (número)

29 de Março de 2007

Resumo

Este documento apresenta uma possível estrutura e conteúdo que deverá ter um relatório de um projecto do 1º ano de uma das licenciaturas de Informática leccionadas pelo Dep. Informática da UM. Serve simultaneamente de exemplo de relatório e de manual resumido de instruções para a sua elaboração. Os relatórios deste tipo têm uma dimensão entre 4 e 8 páginas, para além de eventuais anexos. Descreve-se neste relatório a análise e especificação da funcionalidade do banco de registos de um CPU (inspirado no IA-32), incluindo os algoritmos e as estruturas de dados indispensáveis, bem como o tipo de testes de validação a realizar.

Conteúdo

1 Introdução

Uma introdução começa normalmente com a apresentação do problema que se pretende resolver e sua contextualização (o "enquadramento").

No caso deste relatório, é um documento de treino das capacidades de comunicação escrita dos estudantes do 1º ano de informática da UM, que tem também como objectivo relatar as diversas fases por que o grupo de trabalho teve de passar para chegar ao resultado final na resolução de um dado problema, apresentado como o "Projecto". Este relatório faz parte integrante do projecto do par de disciplinas PI+AC (LCC/LEI, com a designação de "Projecto Integrado").

Uma vez elaborado, em LaTeX, o relatório e respectivos anexos (se existirem) deverão ser compilados para PDF. O conjunto de ficheiros resultantes, LaTeX, PDF e os ficheiros de código e respectivos dados (de entrada e/ou de teste) deverão ser colocados numa pasta e esta compactada (formato ZIP). O ficheiro resultante deverá então ser submetido electronicamente em `http://nirvana.di.uminho.pt/labdotnet/Submit_PI_0607`. Se alguém tiver que submeter o seu trabalho enquanto membro de mais que um grupo (um para PI, outro para AC), deverá contactar directamente o docente de AC.

A introdução termina normalmente com uma apresentação da estrutura do relatório, e o que aqui se apresenta é apenas um exemplo para a fase 1 do projecto. A Secção 2 descreve e analisa o problem a resolver, especificando ainda os seus requisitos. A Secção 3 propõe uma solução para o problema, em termos de algoritmos e estruturas de dados a usar, enquanto a Secção seguinte descreve sumariamente algumas alternativas e decisões pertinentes na codificação da resolução e consequentes testes de validação a efectuar. O relatório termina com uma Secção de Conclusões, onde se discutem de uma forma crítica os resultados gerais obtidos com a elaboração do trabalho.

Comentários adicionais: o Resumo deverá apenas conter uma descrição resumida do conteúdo do relatório, e não mais que 1/3 deverá ser usado para contextualizar o problema; se forem consultadas obras bibliográficas que merecem ser referidas ou sítios na Web que merecem algum destaque, deverá ser criada uma nova Secção "Referências", não numerada, após as "Conclusões", contendo uma lista numerada dessas referências, ordenada pela ordem com que surgem no texto.

2 Análise e Especificação

Pretende-se especificar o banco de registos (BR) que contém duas operações:

- operação de leitura do conteúdo de um registo (de 8 ou de 32 *bits*; em que o de 8 é uma parte de um de 32 *bits*);
- operação de escrita (alteração do conteúdo) num registo (de 8 ou de 32 *bits*; e o de 8 é uma parte de um de 32 *bits*).

2.1 Descrição informal do problema

O banco de registos contempla vários registos. Destes, alguns são de uso genérico, tais como o ESI, EDI, EAX, EBX, ECX e EDX com as suas versões correspondentes de 8 *bits*, AL, BL, CL e DL (*bytes* menos significativos de EAX, EBX, ECX e EDX respectivamente) e AH, BH, CH e DH (segundos *bytes* menos significativos, contíguos aos anteriores). Outros registos, tais como o ESP, EBP, e EIP são específicos (*stack pointer*, *base pointer* e *instruction pointer* respectivamente).

2.2 Especificação dos Requisitos

Pretende-se especificar as operações de leitura e escrita destes registos tendo em conta as seguintes restrições:

- os registos EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI e EIP são registos de 32 *bits*;
- os registos AL, BL, CL e DL são de 8 *bits* e representam o *byte* menos significativo dos registos EAX, EBX, ECX e EDX respectivamente, caso uma das versões seja modificada, tal deverá repercutir-se na outra;
- a restrição anterior deverá existir também para os registos AH, BH, CH e DH que representam o segundo *byte* menos significativo dos registos EAX, EBX, ECX e EDX respectivamente.

O registo de Flags não é considerado nesta especificação preliminar, ficando adstrito à ALU.

2.2.1 Dados

Os dados do problema são:

- designação dos registos a aceder;
- indicação do valor a ser escrito num dado registo;
- a indicação da operação a efectuar, se leitura, se escrita;

Em termos de implementação decidiu-se implementar duas funções em vez de apenas uma e assim prescindindo da indicação da operação a efectuar.

2.2.2 Pedidos

Os valores pedidos são os seguintes:

- a indicação de operação efectuada com sucesso ou não (para o caso de se ter fornecido um valor inadequado, como por exemplo um registo que não existe);
- o valor do registo que se pediu para ser lido.

2.2.3 Relações

Neste caso não existem relações. Estas fazem sentido no caso de uma unidade que se relacione com outras como por exemplo a unidade de controlo com a ALU.

3 Concepção/desenho da Resolução

3.1 Estruturas de Dados

A unidade Banco de Registos é essencialmente constituída por uma estrutura de dados que represente o conjunto dos 9 registos de 32 bits. Estes são armazenados num array de inteiros com 9 elementos, em que cada uma dos elementos representa um dos registos de 32 bits (utilizando o tipo inteiro) por esta ordem: EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI e EIP. Os registos de 8-bits serão parte de alguns desses registos de 32 bits, tal como especificado pela Intel e referido anteriormente. Este aspecto deverá ser devidamente considerado no manuseamento do conteúdo desses registos.

3.2 Algoritmos

Existem duas funções principais a implementar, e várias secundárias de apoio às primeiras. As funções principais que se propõem para o módulo do Banco de Registos são as seguintes, seguidas do respectivo protótipo em C para explicitar de modo claro os argumentos a passar e receber:

- função para ler o conteúdo de um registo;
  protótipo: `int ler_registo(registo reg);`
- função para escrever um valor de 32-bits num registo;
  protótipo: `void escrever_registo(registo reg, int valor)`.

Para além destas funções, as seguintes poderiam ajudar a implementação destas de leitura e escrita:

- como não existe nenhum argumento que especifique o tamanho do registo, e em operações de escrita de um valor de 8 bits este será passado à função na posição dos 8 bits menos significativos do segundo argumento (o valor do tipo int), conviria ter uma função que recebesse como parâmetro o registo e devolvesse "verdadeiro" caso este seja de 32 bits ou "falso" caso contrário; protótipo: `char e_reg32(registo reg)`;
- uma função que nos diga se um dado registo de 8 bits corresponde ao byte menos significativo ou não; esta função só deverá ser aplicada a registos de 8 bits, e devolverá "verdadeiro" caso o registo corresponda ao byte menos significativo; protótipo: `char e_menos_significativo(registo reg)`;
- uma função que retorne o registo de 32 bits correspondente ao de 8 bits que queremos; protótipo: `registo reg_correspondente(registo reg)`.

Nestas condições, então a função `ler_registo` deverá funcionar da seguinte forma:

- se o registo for de 32 *bits*, devolver o valor;
- se o registo for de 8 *bits* e corresponder ao *byte* menos significativo, devolver o *byte* menos significativo do registo correspondente de 32 *bits*;
- se o registo for de 8 *bits* e corresponder ao segundo *byte* menos significativo, devolver o segundo *byte* do registo correspondente de 32 *bits*.

A função `escrever_registo` funciona da seguinte forma:

- se o registo for de 32 *bits*, modifica o seu valor;
- se o registo for de 8 *bits* transformar o valor recebido (de 32 *bits*) num valor de 8 *bits*;
- se o registo for de 8 *bits* e corresponder ao *byte* menos significativo:
  - aplicar uma máscara para remover o valor do *byte* menos significativo do registo de 32 *bits* correspondente;
  - colocar no *byte* menos significativo o valor de 8 *bits* passado à função.
- se o registo for de 8 *bits* e corresponder ao segundo *byte* menos significativo:
  - aplicar uma máscara para apagar o valor do segundo *byte* menos significativo do registo correspondente de 32 *bits*;
  - colocar o valor de 8 *bits* passado à função na posição correcta.

4 Codificação e Testes

4.1 Alternativas e Decisões Pertinentes

Nesta secção vão ser apresentadas algumas sugestões de codificação da estrutura de dados que representa os registos, e de algoritmos de manipulação de máscaras que permitem manusear blocos de bits no conteúdo de um registo de 32 bits.

Os registos são armazenados num *array* com 9 posições, em que cada uma das posições representa um dos registos de 32 *bits* (utilizando o tipo inteiro) por esta ordem: EAX , EBX , ECX , EDX , ESP , EBP , ESI , EDI e EIP. Em termos de código C, isso equivale à instrução:

```
int valor_registo[9];
```

Para referenciar os registos, poder-se-ia adoptar um valor inteiro e codificar cada um deles como uma macro que seria tratada pelo pré-processador do C. Neste caso, o código resultante para os primeiros três registos seria algo do género:

```
#define EAX 0
#define EBX 1
#define ECX 2
```

Contudo, existe uma maneira mais simples de especificar todos os registos e automaticamente associar a cada um deles um número inteiro que poderá depois ser utilizado para indexar o *array* correspondente. Para isso utilizamos um tipo enumerado que nos confere essa possibilidade. A definição do tipo de dados `registo` seria a seguinte:

```
typedef enum{EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI, EIP,
AL, BL, CL, DL, AH, BH, CH, DH, ERRO} registo;
```

Utilizando este sistema, torna-se fácil definir as funções auxiliares. Por exemplo, para ver se um registo é de 32 *bits* só é necessário ver se o seu valor numérico é menor ou igual ao de EIP. Para ver se um registo de 8 *bits* se refere ao *byte* menos significativo, só é necessário ver se o valor que o idenfifica se encontra entre AL e DL.

Para extrair o *byte* menos significativo de um valor procede-se da seguinte forma:

- aplicar uma máscara para retirar os 3 *bytes* mais significativos e deixar apenas o menos significativo;
- retornar esse valor.

Para extrair o segundo *byte* de um valor procede-se da seguinte forma:

- aplicar uma máscara para retirar os 2 *bytes* mais significativos e o menos significativo;
- efectuar um deslocamento para a direita de 8 *bits* para que o valor a retornar esteja na grandeza correcta;
- retornar esse valor.

A aplicação de máscaras para remover *bits* de um valor é efectuada mediante operações lógicas do tipo *e* através do operador `&`. Para isso é necessário colocar na máscara a zero os *bits* que se pretendem remover do valor e a um os que se pretendem manter. Assim, para remover os três *bytes* mais significativos, utiliza-se a máscara `0x000000ff`; caso se pretenda remover os dois *bytes* mais significativos e o menos significativo, usa-se a máscara `0x0000ff00`.

O deslocamento para a direita utiliza o operador `>>` enquanto que o deslocamento para a esquerda usa o operador `<<`. Estas operações são necessárias para deslocar um valor que se encontra no segundo byte para o primeiro ou vice-versa.

Para se colocar um valor de 8 *bits* num dos *bytes* de um valor de 32 *bits* é necessário começar por remover o valor que existia no *byte* correspondente e depois utilizando um *ou lógico* para colocar lá o valor de 8 *bits* que se pretende. Repare que o *ou lógico* de um 1 coloca o *bit* a 1 enquanto que o ou lógico de um 0 é o elemento neutro da operação.

4.2 Testes de Validação

Os testes de validação deverão garantir a correcção das funções implementadas em todos os casos. Devem-se testar não só as condições normais de funcionamento e as as condições limite bem como a introdução de dados inválidos (quando viável). Assim deverão testar-se os seguintes casos:

- garantir que a função `e_reg32` funciona correctamente;
- garantir que a função `e_mais_significativo` funciona correctamente;
- garantir que a função `reg_correspondente` funciona correctamente;

- escrever um valor num registo de 32 *bits* e depois ler o valor do mesmo registo e garantir que ambos os valores são iguais;
- escrever um valor num registo de 8 *bits* correspondente à parte menos significativa e depois ler o valor do mesmo registo e garantir que ambos os valores são iguais;
- escrever um valor num registo de 8 *bits* correspondente à parte contígua à menos significativa e depois ler o valor do mesmo registo e garantir que ambos os valores são iguais;
- escrever um valor maior do que 8 *bits* num registo de 8 *bits* e garantir que só foram modificados os 8 *bits* correctos;
- escrever um valor num registo de 32 *bits* e depois ler os registos correspondentes de 8 *bits* e verificar que os valores são os correctos; e.g., escrever `0x0fea1de4` em `EAX` e garantir que o valor em `AH` é `0x1d` e em `AL` é `0xe4`;
- ler um valor de um registo de 32 *bits* e alterar o registo correspondente ao *byte* menos significativo, ler novamente o registo de 32 *bits* e garantir que o valor se modificou correctamente;
- ler um valor de um registo de 32 *bits* e alterar o registo correspondente ao segundo *byte* menos significativo, ler novamente o registo de 32 *bits* e garantir que o valor se modificou correctamente.

5 Conclusões

Síntese do Documento.
Estado final do projecto. Análise crítica dos resultados.
Sugestões para trabalho futuro.

A Código do Programa

Esta fase do projecto não tem qualquer código a desenvolver, pelo que este anexo não deverá existir na Fase 1 da parte de SC. Mas se houvessse código a apresentar, ele seria assim disponibilizado (exemplo do código do hello.c):

```
#include <stdio.h>

int main(void) {
    printf("Hello World\n");
    return 0;
}
```