MC404 Organização de Computadores
e Linguagem de Montagem

IC – Unicamp

Segundo Trabalho – 2s2008

David Burth Kurka 070589
Felipe Eltermann Braga 070803

Neste laboratório, a proposta foi implementar, em linguagem de montagem, um desmontador; isto é, um programa que interprete os bytes de um arquivo binário e produza o código que o gerou.

Para isso, a idéia fundamental do funcionamento do programa é, em alto nível:

- Carregar o arquivo binário inteiro na memória;
- Abrir um arquivo de saída (que conterá o código) e escrever nele as diretivas iniciais do código;
- Interpretar cada instrução do arquivo binário e escrever, uma a uma, no arquivo de saída, até o fim do arquivo executável;
- Fechar o arquivo de saída.

A implementação da parte de manipulação dos arquivos e da memória não trouxe novidades com relação ao primeiro projeto:
– O arquivo de entrada é aberto, seu apontador interno é movido até o final (a fim de encontrarmos seu
tamanho em bytes), é lido inteiro para o segmento extra de dados (ES) e fechado.
– O arquivo de saída é criado, são escritas as diretrizes iniciais, é realizada a interpretação dos bytes do arquivo binário e as instruções são montadas uma a uma num vetor 'linha_de_comando', o vetor é escrito no arquivo de saída, repetese a interpretação até o fim do arquivo binário, o arquivo é fechado.

Dessa forma, a parte complicada foi a interpretação dos bytes:

Interpretação do código binário

Pensamos, inicialmente, em implementar uma árvore binária de desvios, de modo a ir "montando" a linha de comando (cada bit do byte seria um nó, e dependendo do próximo bit, a execução desvia para um ou outro lugar). Dessa forma, poderíamos ter na memória listas de diretrizes, instruções, registradores, e iríamos apenas concatenando as informações em um vetor.

No entanto, seriam necessárias muitas comparações para chegar à instrução, sem contar o acesso à memória (seria necessário buscar as palavras na memória, trazer a algum registrador [logo, de 2 em 2 caracteres] e então escrevêlas novamente na memória [em linha_de_comando]).

Portanto, decidimos implementar uma tabela de desvios em que o índice seria duas vezes o valor do próprio byte a ser interpretado, conseguindo encontrar rapidamente a funcao correspondente ao opcode do byte.

Assim, não ocorre retardo de busca, o acesso é direto à instrução e o vetor 'linha_de_comando' é montado de acordo com cada instrução.

A interpretação dos opcodes do arquivo .com é feita bit por bit, até encontrar a instrução referente à linha "int 00020h" em nasm. Quando essa instrução é encontrada, os demais bits do arquivo de entrada são considerados como pertencentes ao segmento de dados e armazenados dessa forma no arquivo .asm gerado.

Operações com opcodes:

Por falta de tempo, não conseguimos implementar todas as instruções do 8086.

Das que implementamos, as que apresentaram maior dificuldade foram as que envolviam o endereçamento em memória, já que foi necessário avaliar 2 tipos genéricos:

– r/m como destino (ex: mov reg8/mem8, reg8)

– r/m como fonte (ex: mov reg8, reg8/mem8)

E para cada tipo, considerar os casos de 8 bits e de 16 bits.

Para esta finalidade, foram implementadas funções auxiliares do tipo RM[Fonte/Destino] [8/16], cuja especificação em baixo nível está nos comentários do código.

Observação: O projeto foi escrito para o montador NASM.