João Pedro Garcia - 1211768

Matheus Caldas - ???????

Relatorio – Geração de Código.

O objetivo inicial do trabalho é desenvolver, um compilador para a linguagem *mínima* baseado em um programa em C. A função principal “geracod” lê um arquivo texto(.txt) contendo o código fonte da linguagem mínima a ser lido. Lendo esse arquivo atribuímos os respectivos códigos de maquina da funções escritas em mínima para gerar o programa em Assembly.

Primeiramente alocamos um grande vetor que irá conter todas as instruções do código de maquina. Durante o processo de construção imprimimos os valores para teste e assim fomos conferindo usando Dumps em programas equivalentes para que tivéssemos certeza do que estávamos fazendo. Dessa forma conferimos a maior parte dos erros.

O programa utiliza 3 grandes funções para a identificação e tradução das funções em mínima para código de maquina. Para todos os casos lemos as linhas do programa e de acordo com que instrução ela começa sabemos com qual função devemos trabalhar. Caso tenhamos uma variável ou parâmetro direto sabemos que será necessariamente uma função de operação. Caso tenhamos a palavra ifeq sabemos que é uma função de desvio e ret uma chamada de retorno.

O primeiro tipo é a função de operação, podendo ser adição, subtração ou multiplicação. Para essa primeira função a chave é a obtenção do sinal. Houve grande dificuldade aqui por conta disso. O membros da operação podem ser constantes sinalizados com $ na frente, parâmetros já estabelecidos ou variável. Caso o primeiro membro da linha seja uma variável devemos colocar esse valor em uma pilha.

“ifs” e “elses” separam o código para cada caso. Utilizamos para todas as operações ecx e edx. A grande dificuldade foi que estávamos utilizando subl de modo errado pois ele atribui o resultado ao segundo elemento(ex: subl ecx edx atribuirá o resltado a edx) e dávamos mov de maneira errada pois nos confundimos e não pensamos nisso no inicio. Quando alternávamos o mov sempre obtiamos o resultado errado, ou o valor oposto do resultado ou o valor de um dos dois elementos da operação. Após algumas tentativas descobrimos o erro e resolvemos consertando o movl também. Nos confundíamos também por que Add e Imul vinham funcionando sem problemas, mas é por que a ordem dos fatores e das parcela não altera o produto ou resultado , já na subtraçãoo isso não é verdade.

A função de retorno funcionou bem desde o principio. Bastava mover o resultado desejado para eax. Mas tínhamos de lidar com três tipos de retorno. No caso de variável ou parâmetro bastava apenas a atribuição. O mais complexo era o retorno de uma constante no qual era necessário se trabalhar com shifts por questões de leitura, little endian e big endian.

O mais complexo foi a função ifseq. Caso a linha começasse com esse comando deveríamos comparar os dois valores e caso fossem iguais pular para a linha sinalizada pelo ultimo valor. Como não se sabia o numero total de comandos até a linha desejada tínhamos de reservar espaço no vetor em que guardávamos o código e sinalizar depois o endereço do comando para onde o programa deveria pular.

Para tal foi necessário criar vetores e variáveis auxiliares que nos ajudavam a guardar os endereços dos comandos e saber em que linha estávamos, assim poderíamos efetuar o jump para o local desejado. Uma grande dificuldade foi no momento de preencher o vão dos 4 bytes que continha o endereço de para onde o jump iria. Estávamos fazendo uma atribuição errada de variável. Ao invés de passar o valor contido no vetor passamos o endereço dele. Logo em seguida também tivemos um segmentation fault, core dumped que levou um bom tempo para se resolver pois em um detalhe colocamos uma subtração no local errado.

O programa conta com algumas funções auxiliares como verifica\_var\_par\_cte que lê se o elemento é uma constante, variável ou parâmetro para que em outras funções saibamos com o que estamos lidando e como devemos trabalhar com ele. verifica\_operacao é outra função utilizada para o caso de operação onde vemos se estamos lidando com subtração, adição ou multiplicação.

O programa parece rodar perfeitamente em todo tipo de maquina. Após a finalização dele percebemos que manuseávamos as variáveis de modo que funcionaria em apenas algum computadores por questão de alocação e refizemos o esquema de alocar na pilha usando movl de forma correta, sendo assim o programa roda para compilar qualquer programa de linguagem mínima dentro desses casos, fazendo até mesmo loops. O compilador porém não aceita nenhum tipo de equivoco no arquivo de entrada. Caso cada elemento da linha não esteja separado por espaços ou haja algo que não está definido entre os comandos exigidos no enunciado do trabalho ele não irá rodar direito.