Tokuwaga
Compilador de la Máquina Sencilla

Diseño de Sistemas con FPGA
Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Primer Cuatrimestre, 2016

Abstract

Implementación de un compilador para la máquina sencilla, añadir funciones al repertorio de la máquina mediante pseudo instrucciones, optimización del uso de memoria.

Palabras claves: compilador, pseudo-instrucciones, lenguaje ensamblador, lenguaje máquina, parser, desensamblador.

Ishikame, Emiliano - xxx/xx - emilianoishikame@yahoo.com.ar
Torrico, Myrko - xxx/xx -mirko.torrico@gmail.com

Índice

1. Introducción

Nuestro ensamblador, Tokuwaga, se encarga de recibir un archivo.asm y generar su transformacin a lenguaje de mquina. El archivo recibido pasa por el siguiente proceso:

Limpia el código: se deshace de comentarios y reconoce variables, constantes y etiquetas para poder direccionarlas después, y hace el reemplazo de las pseudo-instrucciones. Optimiza Memoria. Parsea el código.

En el inicio, Tokuwaga soportaba sólo las instrucciones de nuestra Máquina sencilla (MOV,ADD,CMP, etc). Luego para facilitar la programación en assembler, nos vimos obligados a aceptar pseudo-instrucciones como: JUMP, SUB, LEA, entre otros.

Finalmente Tokuwaga permite el uso de constantes, usando como prefijo @; variables y etiquetas de la pinta etiqueta:. Para el testeo también se desarrolló un desensamblador, el cual transforma el lenguaje máquina en código ensamblador.

2. Operaciones

Lenguaje del miniCompilador Instrucciones válidas de ensamblador

2.1. DEFINICIÓN DE TIPOS

P	$DEVICE\_DIR \vert REG\_SEL$
D	Direccion Destino
S	Direccion Fuente
A	ADDR
@12	constante (DECIMAL)

2.2. OPERACIONES ARIMÉTICO/LÓGICAS

OPCODE	Función	Descripción
ADD S,D	[D]=[S]+[D],Z=([S]+[D])==0	Suma S+D y lo almacena en D, Si S+D=0 entonces flag Z es 1 si no flag Z=0.
CMP S,D	Z=([S]-[D])==0	Si S-D=0 , entonces el flag Z=1 si no flag Z = 0.
DEC D	[D]=[D]-1	Decrementa la variable D en 1.
INC D	[D]=[D]-1	Incrementa la variable D en 1 (PSEUDO).
SUB D,S	[D]=[D]-[S]	Resta S a D y lo guarda en D.
SHIFTL A,S	[A]=A>>S	Aplica un shift lógico a izquierda sobre A tantos bits como indica S.
SHIFTR A,S	[A]=A<<s	Aplica un shift lógico a derecha sobre A tantos bits como indica S.

2.3. MOVIMIENTO DE DATOS

OPCODE	Función	Descripción
MOV S,D	[D]=[S]	Mueve lo que esta en la posición de memoria S a la posición D.
MOV S,[D]	[D]=[[S]]	Mueve lo que esta en la posición de memoria apuntada por S a la posición de memoria D.
MOV S,[D]	[[D]]=[S]	Mueve lo que esta en la posición de memoria S a la posición apuntada por D.
MOV [S],[D]	[[D]]=[[S]]	Mueve lo que esta en la posición apuntada por S a la posición apuntada por D.
LEA "TAG",D	[D]="TAG"	Carga la dirección en que se encuentra la etiqueta,variable o constante "TAG" en la posición D.

2.4. OPERACIÓN DE ENTRADA/SALIDA

OPCODE	Función	Descripción
IN P,D	[D]=[S]	Mueve a la posición de memoria D lo que esta en el puerto P.
OUT P,S	[D]=[[S]]	Mueve al puerto P lo que esta en la memoria en la posición S.

2.5. OPERACIONES DE SALTO

OPCODE	Función	Descripción
BEQ "TAG"	Z==1 → PC="TAG"	salta si el flag Z esta activo a la posición de la etiqueta o posición "TAG".
CALL "TAG"	[[reg]=PC, PC="TAG", reg=reg-1	Llama a la función "TAG".
RET	reg+1, PC=[reg]	Retorna de un call.
JMP "TAG"	PC="TAG"	Salto a la posición de la etiqueta o posición "TAG.

2.6. DEFINICIÓN DE DATOS

OPCODE	Descripción
DW xxx	Define un word de 16 bits en esa posición de memoria (el número ingresado debe ser decimal).
LABEL:	Define una etiqueta.

Soporta uso de variables y constantes

3. Instrucciones en orden alfabético

3.1. Instrucción ADD

El Destino de la función ADD no puede ser una constante.

OPCODE	Descripción
ADD a,b	Suma la variable a con b y lo guarda en b.
ADD a,1	Suma la posición 1 con a y lo guarda en la posición 1.
ADD 1,a	Suma la variable a con 1 y lo guarda en la variable a.
ADD @1,a	Suma la variable a con la constante y lo guarda en la variable a.

3.2. Instrucción BEQ

OPCODE	Descripción
BEQ 1	si la comparación FZ=1 salta a la posición 1.
BEQ LABEL	si la comparación FZ=1 salta a la etiqueta LABEL.

3.3. Instrucción CALL

OPCODE	Descripción
CALL LABEL	Salta a la etiqueta LABEL.
CALL 100	Salta a la posición 100 (valor absoluto).

3.4. Instrucción CMP

OPCODE	Descripción
CMP a,b	Compara las variables a y b y guarda el resultado en el flag Z.
CMP a,1	Compara a y con la posicion de memoria 1 y guarda el resultado en el flag Z.
CMP 1,2	Compara la posición de memoria 1 y 2 y guarda el resultado en el flag Z.
CMP @1,a	Compara una constante con la variable a y guarda el resultado en el flag Z.

3.5. Instrucción DEC

El Destino de la función DEC no puede ser una constante.

OPCODE	Descripción
DEC a	incrementa la variable a en 1.
DEC 0	incrementa lo que hay en la posición 0 en 1.

3.6. Instrucción DW

Define una constante.

3.7. Instrucción INC

El Destino de la función INC no puede ser una constante.

OPCODE	Descripción
INC a	incrementa la variable a en 1.
INC 0	incrementa lo que hay en la posición 0 en 1.

3.8. Instrucción IN/OUT

El Destino de la función IN no puede ser una constante.

OPCODE	Descripción
IN puerto,10	mueve el puerto a la posición de memoria 10.
IN puerto,D	mueve el puerto a la variable S.
OUT puerto,10	mueve la posición de memoria 10 al puerto.
OUT puerto,S	mueve la variable S al puerto.
OUT puerto,@10	mueve una constante al puerto.

El puerto tiene direcciones validas de 0 a 31. Para los dispositivos predefinidos hay etiquetas para facilitar la identificación.

- LED
- PUERTO_0_SHIFTER
- PUERTO_1_SHIFTER
- PUERTO_2_SHIFTER
- PUERTO_3_SHIFTER
- UART_TX
- UART_RX
- UART_FULL
- UART_EMPTY
- SSEG
- BTNS
- SWT

3.9. Instrucción JMP

El Destino de la función IN no puede ser una constante.

OPCODE	Descripción
JMP label	salta a la etiqueta label.
JMP 10	mueve el puerto a la variable S.

3.10. Instrucción LEA

Permite obtener la dirección donde el ensamblador almacena una variable.

OPCODE	Descripción
LEA LABEL,A	Carga la posición de la etiqueta LABEL en A.
LEA b,A	Carga la posición de la variable b en A.
LEA @100,A	Carga la posición de la constante 100 a A.

3.11. Instrucción MOV

El Destino de la función MOV no puede ser una constante.
Si el parámetro pasado por referencia esta fuera de los límites de memoria, la función no garantiza su ejecución y/o integridad de la memoria.

OPCODE	Descripción
MOV S,D	Mueve de S a D..
MOV S,1	Mueve lo que esta en la "variable" S a la posición 1.
MOV 1,D	Mueve lo que esta en la posicion 1 a la variable D.
MOV @10,S	Mueve una constante a la variable S.
MOV @10,1	Mueve una constante a la posición de memoria 1.
MOV [S],1	Mueve lo que esta en la posición referenciada por S a la posición 1.
MOV [1],D	Mueve lo que esta en la posición referenciada por la posición 1 a la variable D.
MOV @10,[D]	Mueve una constante a la posición referenciada por D.
MOV @10,[1]	Mueve una constante a la posición referenciada por la posición 1.
MOV @10,[@14]	Mueve una constante a la posición referenciada por la constante (equivale a MOV @10,14).
MOV S,[@14]	Mueve lo que esta en la "variable" S a la posición referenciada por la constante (equivale a MOV @10,14).

3.12. Instrucción RET

Retorna de la ejecucion de un call.

3.13. Instrucción SHIFT

El Destino de la función SHIFT no puede ser una constante.

OPCODE	Descripción
SHIFTR X,10	Desplaza X lógicamente a derecha tantos bits como indica la posición de memoria 10.
SHIFTR X,S	Desplaza X lógicamente a derecha tantos bits como indica la variable S.
SHIFTR X,@10	Desplaza X lógicamente a derecha tantos bits como indica la constante.
SHIFTL X,10	Desplaza X lógicamente a izquierda tantos bits como indica la posición de memoria 10.
SHIFTL X,S	Desplaza X lógicamente a izquierda tantos bits como indica la variable S.
SHIFTL X,@10	Desplaza X lógicamente a izquierda tantos bits como indica la constante.
SHIFTR 25,10	Desplaza lo que hay en la posición 25 lógicamente a derecha tantos bits como indica la posición de memoria 10.
SHIFTR 25,S	Desplaza lo que hay en la posición 25 lógicamente a derecha tantos bits como indica la variable S.
SHIFTR 25,@10	Desplaza lo que hay en la posición 25 lógicamente a derecha tantos bits como indica la constante.
SHIFTL 25,10	Desplaza lo que hay en la posición 25 lógicamente a izquierda tantos bits como indica la posición de memoria 10.
SHIFTL 25,S	Desplaza lo que hay en la posición 25 lógicamente a izquierda tantos bits como indica la variable S.
SHIFTL 25,@10	Desplaza lo que hay en la posición 25 lógicamente a izquierda tantos bits como indica la constante.

Los valores insertados son siempre 0, el máximo valor de desplazamiento es 15. Si se excede este límite la función puede no devolver un valor o provocar alteraciones en la memoria.

3.14. Instrucción SUB

El Destino de la función SUB no puede ser una constante.

OPCODE	Descripción
SUB X,Y	Resta la variable Y a la variable X y lo guarda en X.
SUB X,1	Resta el valor que hay en la posición 1 a la variable X y lo guarda en X.
SUB X,@1	Resta la constante 1 a X y lo guarda en X.
SUB 1,X	Resta la variable X al valor que hay en la posición 1 y lo guarda en la posición 1.
SUB 1,@1	Resta la constante 1 al valor que hay en la posición 1 y lo guarda en la posición 1.
SUB 1,12	Resta el valor que hay en la posición 12 al valor que hay en la posición 1 y lo guarda en la posición 1.

4. Pseudo-Instrucción

4.1. Introducción

Dado que la máquina tiene pocas instrucciones disponibles, para facilitar el trabajo al programador, se decide proveer de estas instrucciones. Esto no significa que sea necesariamente mejor que la que pueda implementar. Brinda este servicio de forma general, por lo que puede no ser óptimo.

4.2. Desarrollo

4.2.1. INC

La funcion INC se implementa mediante la creación de la constante 1 y utilizando la instrucción ADD. De modo que INC x se transforma en ADD @1,X

```
INC X   =       ADD c,X
            c:  DW 1
```

4.2.2. DEC

La funcion INC se implementa mediante la creación de la constante 65535 y utilizando la instrucción ADD. De modo que INC x se transforma en ADD @1,X

```
DEC X   =       ADD c,X
            c:  DW 65535
```

4.2.3. SUB

La funcion SUB se implementa mediante una resta sucesiva hasta que el segundo operando es cero, luego se mueve el resultado al segundo operando.

```
SUB A,B =           MOV B,Var
            loop:   CMP @0,Var
                    BEQ fin
                    ADD @65535,Var
                    ADD @65535,A
                    CMP 0,0
                    BEQ loop
            fin:
```

4.2.4. LEA

La funcion LEA se implementa mediante la creación de la constante c, la cual es la posición donde se encuentra X y utilizando la instrucción MOV. De modo que LEA x,0 se transforma en MOV c,X

```
LEA c,X     =   MOV a,X
                a:posicion de memoria de c asignada por el compilador
```

4.2.5. MOV

La funcion MOV con indirección se implementa mediante un algoritmo que modifique la memoria para que la siguiente instrucción que se ejecute sea un MOV con indireccion.
MOV [A],B

```
            MOV A,X
            MOV @0,Contador
    loop:   CMP CONTADOR_LOOP,7
            BEQ FIN
            ADD X,X
            ADD @65535,Contador
            CMP 0,0
            BEQ loop
    FIN:    ADD @32768,X
            LEA B,var
            ADD var,X
    X:      MOV A,0
```

MOV [A],[B]

```
            MOV A,X
            MOV @0,Contador Loop
    loop:   CMP CONTADOR_LOOP,7
            BEQ FIN
            ADD X,X
            INC contador_loop
            CMP 0,0
            BEQ loop
```

```
FIN     ADD @32768,X
        ADD B,X
X:      MOV A,0
```

MOV A,[B]

```
        LEA A,X
        MOV @0,Contador
loop:   CMP CONTADOR_LOOP,7
        BEQ FIN
        ADD X,X
        ADD @65535,Contador
        CMP 0,0
        BEQ loop
FIN:    ADD @32768,X
        LEA B,var
        ADD var,X
X:      MOV A,0
```

Aprovechado los modulos de E/S, se puede simplificar la instrucción MOV [A],B

```
        OUT PUERTO_0_SHIFTER,@7
        OUT PUERTO_1_SHIFTER,A
        IN  PUERTO_2_SHIFTER,X
        ADD @32768,X
        LEA B,var
        ADD var,X
X:      MOV A,0
```

MOV [A],[B]

```
        OUT PUERTO_0_SHIFTER,@7
        OUT PUERTO_1_SHIFTER,A
        IN  PUERTO_2_SHIFTER,X
        ADD @32768,X
        ADD B,X
X:      MOV A,0
```

MOV A,[B]

```
        LEA A,X
        OUT PUERTO_0_SHIFTER,@7
        OUT PUERTO_1_SHIFTER,X
        IN  PUERTO_2_SHIFTER,X
        ADD @32768,X
        ADD B,X
X:      MOV A,0
```

```
\subsubsection{SHIFT}
La funcion SHIFT era parcialmente soportada hasta que fue a\~nadido el m\'odulo de
entrada/salida que permite dicha operaci\'on.

SHIFTR X,@const
\begin{verbatim}
    OUT PUERTO_0_SHIFTER,@const
    OUT PUERTO_1_SHIFTER,X
    IN  PUERTO_3_SHIFTER,X
```

SHIFTR X,Y

```
    OUT PUERTO_0_SHIFTER,Y
    OUT PUERTO_1_SHIFTER,X
    IN  PUERTO_3_SHIFTER,X
```

SHIFTL X,@const

```
    OUT PUERTO_0_SHIFTER,@const
    OUT PUERTO_1_SHIFTER,X
    IN  PUERTO_2_SHIFTER,X
```

SHIFTL X,Y

```
    OUT PUERTO_0_SHIFTER,Y
    OUT PUERTO_1_SHIFTER,X
    IN  PUERTO_2_SHIFTER,X
```

5. Compilación

El método empleado para compilar el programa es el escaneo en 2 pasadas. En la primer lectura, se reconocen las etiquetas, variables y constantes y se les asigna un token. Luego, corremos un algoritmo que busca minimizar la cantidad de memoria utilizada por las variables y constantes. Finalmente, en la segunda lectura se procede a reemplazar las líneas por código máquina y para resolver las posiciones desconocidas, utiliza el diccionario creado en el paso anterior.

6. Optimización del uso de memoria

6.1. Motivación

Dado que la máquina tiene memoria finita y, en este caso, muy acotada, se debe aprovechar al máximo este recurso.

6.2. Desarrollo

El problema de memoria se intentá mejorar desde dos puntos de vista: instrucciones y variables. Por el lado de las instrucciones, se busca que las pseudo instrucciones ocupen poca memoria y tengan un rendimiento aceptable. Por el otro lado, se intenta aprovechar la memoria organizando las constantes y variables, de forma que esta sea un conjunto minimal y de menor tamaño posible.

7. Testing

Para testear el código generado por el compilador, se implementó un reensamblador, el cual genera instrucciones (si las existen) a partir del código binario. Dado que las pseudo instrucciones no pueden ser reensambladas y las variables y constantes no son visibles en el código original, el código obtenido es, en general, de mayor tamaño. Además, como algunas pseudo-instrucciones crean la instrucción en el momento de ejecución, se implementó un simulador de la máquina sencilla para poder realizar el seguimiento sobre dichas instrucciones.

8. Mejoras posibles en el futuro

Se puede mejorar la velocidad de compilación mejorando el modo de compilación. Por otro lado, se puede mejorar la distribución de las variables para consumir menos memoria. Por motivos de rápida implementación el compilador, el reesamblador y el simulador fueron creados independientemente, creemos que integrar las tres herramientas en un solo entorno permitira crear código para la máquina sencilla de manera simple. Además, a causa de que el simulador no ofrece la posibilidad que se realicen comandos de entrada, se limita las pruebas de programas a procesamiento estático. Sería idóneo que en el futuro admita esa funcionalidad.

9. NOTAS IMPORTANTES

- Es responsabilidad del programador que las instrucciones de SHIFT,MOV si se ejecutan pasando un parámetro por referencia, esta este dentro del rango válido.
- Las pseudo instrucciones no necesariamente ocupan 1 palabra en memoria.
- Cuando las pseudo-instrucciones son utilizadas, tener cuidado con las referencias absolutas, es recomendable utilizar etiquetas.

Importante: UNA ETIQUETA TIENE PRIORIDAD SOBRE UNA VARIABLE, SI EL NOMBRE DE UNA VARIABLE ES EL MISMO QUE UNA ETIQUETA, CUANDO SE REFERENCIE APUNTARÁ LA ETIQUETA.

10. BUGS

Lista de Known Issues:

- No soporta dos instrucciones válidas en la misma línea.
- Hay una variable de ensamblador que está reservada.
- Si bien hace un chequeo general de los operandos, puede que acepte algo no soportado.