Práctica 6
Paso 2 del ensamblador para SIC extendida
Laboratorio de Programación de Sistemas

Jorge Armando Tovar Ojeda

12 de noviembre de 2015

1. Objetivo

Ensamblar las instrucciones de acuerdo a los formatos y modos de direccionamiento de la SIC extendida y generar el archivo objeto agregando los registros de modificacion correspondientes.

2. Otras secciones

2.1. Explicar el mecanismo utilizado para calcular cada una de las banderas n, i, x, b, p, e y el desplazamiento/direccion.

Para esta seccion se utilizaron enteros para representar las banderas de estado en donde: n,i se tomaron como 2 bits, x,p,e se tomaron como 1 bit y el desplazamiento como 12 bits en caso de el formato 3 y 20 bits en el formato 4 asi como se muestra a continuacion:

Assemblef3 Ensambla el formato 3
func (p *Assembler) Assemblef3(line int, opcode int64, ni int64, x int64, b int64, pp int64, e int64, desp int64)
opcode = opcode | ni (se juntan los ultimos 2 bits de opcode con ni)
asmInst := desp | (e<<12) | (pp <<13) | (b <<14) | (x<<15) | (opcode <<16)
asmInstStr := strconv.FormatInt(asmInst, 16)
asmInstStr = strings.ToUpper(asmInstStr)
asmInstStr = p.formaTo3byte(asmInstStr)
p.Hexcode = append(p.Hexcode, asmInstStr)
p.Addrs = append(p.Addrs, GetTabSim().Progpc[line])

Assemblef4 Ensambla el formato 4
func (p *Assembler) Assemblef4(line int, opcode int64, ni int64, x int64, b int64, pp int64, e int64, desp int64)
opcode = opcode | ni
asmInst := desp | (e <<20) | (pp <<21) | (b <<22) | (x <<23) | (opcode <<24)
asmInstStr := strconv.FormatInt(asmInst, 16)
asmInstStr = strings.ToUpper(asmInstStr)
asmInstStr = p.formaTo4byte(asmInstStr)
p.Hexcode = append(p.Hexcode, asmInstStr)
p.Addrs = append(p.Addrs, GetTabSim().Progpc[line])

Para el caso del desplazamiento se utilizo el algoritmo visto en clase respecto a la relatividad del desplazamiento con respecto al registro CP o al registro B como muestra el siguiente codigo:

AssembleXE ensambla la sic extendida
func (p *Assembler) AssembleXE(line int, nemonic string, indexed int64, symbol string, bRegister int64, ni int64)
ta = GetTabSim().GetSymbolAddrs(symbol)
desp = ta - GetTabSim().Progpc[line]
opcode := p.GetCodeEntry(nemonic)
if desp <= 2047 AND desp >= -2048
if desp ¡0
desp = 4096 + desp
p.Assemblef3(line, opcode, ni, indexed, 0, 1, 0, desp)
else
desp := ta - bRegister
if desp >= 0 AND desp <= 4095
p.Assemblef3(line, opcode, ni, indexed, 1, 0, 0, desp)
else
p.Assemblef3(line, opcode, ni, indexed, 1, 1, 0, 4095)

2.2. Explicar el mecanismo utilizado para almacenar y generar los registros de modificacion.

En este caso los registros de modificacion se almacenan en un arreglo de strings, se generan cuando un formato 4 es ensamblado y el algoritmo implementado fue el siguiente:

Se busca en la tabla de simbolos la direccion del desplazamiento
Si la encuentra significa que se tiene que hacer una nueva entrada de registro relocalizable
Se ensambla segun donde se encuentre el contador de programa + 1 con un desplazamiento de 5 medios bytes y un + (ya que siempre se le va a sumar

mientras no haya terminos relativos negativos)

for key, value = range GetTabSim().table
num, err = strconv.ParseInt(asmInstStr, 16, 0)
num = num AND 0x000FFFFF
if value == num
found = true
if found
dirstr = strings.ToUpper(p.formaTo3byte(strconv.FormatInt(GetTabSim().Progpc[line-1]+1, 16)))
p.mReg = append(p.mReg, "M-dirstr+"05--")
found = false

2.3. Explicar el mecanismo utilizado para soportar la compatibilidad hacia atrás.

Se tomo mucho en cuenta la extension del archivo para realizar la compatibilidad ya que el ensamblado de el direccionamiento simple tiene la misma sintaxis tanto para la SIC estandar como para la SICXE, el siguiente fragmento de codigo muestra como se realizo la validacion de la extension del archivo

simple3:
NEMONICO identificador indexado
if yylex.(*yylexer).firstparse == false AND yylex.(*yylexer).isXE == false
(Ensambla el la SIC ESTANDAR)
else
if yylex.(*yylexer).firstparse == false AND yylex.(*yylexer).isXE == true
if format4 == false
(Ensambla el la SIC XE con formato 3)
else
(Ensambla el la SIC XE con formato 4)

Fragmento de codigo donde se le manda la bandera si es extendida o no
func (p *Parser) Parse(r io.Reader, flag bool, isXE bool) string
l := newLexer(bufio.NewReader(r))
l.firstparse = flag
l.isXE = isXE
yyParse(l)
return l.str

2.4. Describir el procedimiento utilizado para generar el valor hexadecimal para los desplazamientos negativos en instrucciones de formato 3.

Para este procedimiento se realizo el complemento a dos, si el numero es negativo se le suma 4096 de esta forma el numero queda listo para ser ensamblado.
if desp ¡0
desp = 4096 + desp

2.5. Describir los problemas que se hayan presentado durante la práctica y como fueron solucionados.

Se presentaron problemas a la hora de hacer la compatibilidad hacia atras ya que el analizador sintactico confunde entre la SIC estandar y la SICXE, se soluciono haciendo uso de banderas como se describio en la seccion anterior.

2.6. Descripcion de las actividades realizadas por fecha y hora

Fecha Hora	Descripcion
2015-11-05 (0hrs)	Practica creada
2015-11-08 (8hrs)	Se elaboro el mecanismo para manejar las banderas nixbpe y los formatos
2015-11-09 (8hrs)	Se elaboro el codigo objeto
2015-11-12 (3hrs)	detalles y reporte

Cuadro 1: Actividades realizadas

Total de tiempo: 19hrs

3. Conclusiones y posibles mejoras

Existen mejoras que pueden hacerse por ejemplo en la gramatica hay renundancias y por lo tanto el codigo se repite en ciertas partes, se puede realizar una gramatica mas optima, puedo concluir de esta practica demuestra la compatibilidad y la importancia de esta en los sistemas de computo actuales tales como la arquitectura x86 de 32 bits y la x64 de 64 bits de intel
Hora y dia de laboratorio externo: Lunes 7am-8am.

4. Referencias consultadas

Se consulto solamente la documentacion de golang para el uso del mapa y la funcion que convierte a distintas bases.
http://golang.org/pkg/strconv/
https://blog.golang.org/go-maps-in-action