50.054 Pseudo Assembly

ISTD, SUTD

Learning Outcomes

1. Describe the syntax of the source language SIMP.
2. Describe the syntax of the intermediate representation language pseudo-assembly.
3. Describe how pseudo-assembly program is executed.
4. Apply Maximal Munch algorithms to generate a pseudo-assembly code from a given SIMP source code.

Recap

Recall the compiler pipeline

```
[Lexing] -->[Parsing] --> [Semantic Analysis] --> [Optimization]
                                                        |
                                                        ↓
                                              [Target Code Generation]
```

- Parser outputs ASTs.
- An AST is a tree representation of the source language

SIMP

In the rest of this module, we take the SIMP programming language as the subject of study.

$$
\begin{array}{rlll}
(\texttt{Statement}) & S & ::= & X = E; \mid \mathit{return}\ X; \mid \mathit{nop}; \mid \mathit{if}\ \ E\ \{\overline{S}\}\ \mathit{else}\ \{\overline{S}\} \mid \mathit{while}\ E\ \{\overline{S}\} \\
(\texttt{Expression}) & E & ::= & E\ \ OP\ \ E \mid X \mid C \mid (E) \\
(\texttt{Statements}) & \overline{S} & ::= & S \mid S\ \overline{S} \\
(\texttt{Operator}) & OP & ::= & + \mid - \mid * \mid < \mid == \\
(\texttt{Constant}) & C & ::= & 0 \mid 1 \mid 2 \mid \ldots \mid \mathit{true} \mid \mathit{false} \\
(\texttt{Variable}) & X & ::= & a \mid b \mid c \mid d \mid \ldots
\end{array}
$$

SIMP

Example `SIMP1`

```
x = input;
s = 0;
c = 0;
while c < x {
    s = c + s;
    c = c + 1;
}
return s;
```

When `input = 10`, the above should return 45

Intermediate Representation

- We have not fixed a particular target (virtual) machine.
- We want some intermediate representation of the source code that
  - has a smaller set of instructions (compared to the source language).
  - allows us to perform validation and optimization.
  - is flexible for us to generate the target (virtual) code.

Pseudo Assembly

$$
\begin{array}{rcll}
(\texttt{Labeled Instruction}) & li & ::= & l : i \\
(\texttt{Instruction}) & i & ::= & d \leftarrow s \mid d \leftarrow s \ op \ s \mid ret \mid ifn \ s \ goto \ l \mid goto \ l \\
(\texttt{Labeled Instructions}) & lis & ::= & li \mid li \ lis \\
(\texttt{Operand}) & d, s & ::= & r \mid c \mid t \\
(\texttt{Temp Var}) & t & ::= & x \mid y \mid ... \\
(\texttt{Label}) & l & ::= & 1 \mid 2 \mid ... \\
(\texttt{Operator}) & op & ::= & + \mid - \mid * \mid < \mid == \\
(\texttt{Constant}) & c & ::= & 0 \mid 1 \mid 2 \mid ... \\
(\texttt{Register}) & r & ::= & r_{ret} \mid r_1 \mid r_2 \mid ...
\end{array}
$$

- we assume unlimited temperorary variables and registers.
- we use `0` to denote `false` and any `1` constant to denote `true`.
- we disallow constant to be used as the destination of the move and op instructions.
- r_{ret} is a special register for the return statement.
- AKA 3-address code.

Pseudo Assembly

Example PA1

```
1: x <- input
2: s <- 0
3: c <- 0
4: t <- c < x
5: ifn t goto 9
6: s <- c + s
7: c <- c + 1
8: goto 4
9: rret <- s
10: ret
```

How to execute a PA program

PC	Memory	Next
1	{input: 2, x : 2}	2
2	{input: 2, x : 2, s : 0}	3
3	{input: 2, x : 2, s : 0, c : 0}	4
4	{input: 2, x : 2, s : 0, c : 0, t : 1}	5
5	{input: 2, x : 2, s : 0, c : 0, t : 1}	6
6	{input: 2, x : 2, s : 0, c : 0, t : 1}	7
7	{input: 2, x : 2, s : 0, c : 1, t : 1}	8
8	{input: 2, x : 2, s : 0, c : 1, t : 1}	4
4	{input: 2, x : 2, s : 0, c : 1, t : 1}	5
5	{input: 2, x : 2, s : 0, c : 1, t : 1}	6
6	{input: 2, x : 2, s : 1, c : 1, t : 1}	7
7	{input: 2, x : 2, s : 1, c : 2, t : 1}	8
8	{input: 2, x : 2, s : 1, c : 2, t : 1}	4
4	{input: 2, x : 2, s : 1, c : 2, t : 0}	5
5	{input: 2, x : 2, s : 1, c : 2, t : 0}	9
9	{input: 2, x : 2, s : 1, c : 2, t : 0, rret : 1}	10
10	{input: 2, x : 2, s : 1, c : 2, t : 0, rret : 1}	-

```
assuming input = 2
1: x <- input
2: s <- 0
3: c <- 0
4: t <- c < x
5: ifn t goto 9
6: s <- c + s
7: c <- c + 1
8: goto 4
9: rret <- s
10: ret
```

From SIMP to PA - Maximal Munch

Maximal Munch is defined by a deduction system with two rules

- Converting SIMP statements to PA labeled instructions $G_s(S) \vdash lis$
- Converting SIMP expressions to PA labeled instructions $G_a(X)(E) \vdash lis$

$$(\texttt{mAssign}) \quad \frac{G_a(X)(E) \vdash lis}{G_s(X = E) \vdash lis}$$

$$(\texttt{mConst}) \quad \frac{\begin{array}{c} l \; \texttt{is a fresh label} \\ c = conv(C) \end{array}}{G_a(X)(C) \vdash [l : X \leftarrow c]}$$

$$\begin{array}{rcl} conv(true) & = & 1 \\ conv(false) & = & 0 \\ conv(C) & = & C \end{array}$$

e.g. `x = 1` is converted to `1: x <- 1`, assuming label 1 is fresh.

From SIMP to PA - Maximal Munch

Maximal Munch is defined by a deduction system with two rules

- Converting SIMP statements to PA labeled instructions $G_s(S) \vdash lis$
- Converting SIMP expressions to PA labeled instructions $G_a(X)(E) \vdash lis$

$$(\texttt{mAssign}) \quad \frac{G_a(X)(E) \vdash lis}{G_s(X = E) \vdash lis}$$

$$(\texttt{mVar}) \quad \frac{l \texttt{ is a fresh label}}{G_a(X)(Y) \vdash [l : X \leftarrow Y]}$$

e.g. `y = x` is converted to `2: y <- x`, assuming label 2 is fresh.

From SIMP to PA - Maximal Munch

Maximal Munch is defined by a deduction system with two rules

- Converting SIMP statements to PA labeled instructions $G_s(S) \vdash lis$
- Converting SIMP expressions to PA labeled instructions $G_a(X)(E) \vdash lis$

$$
(\mathtt{mAssign}) \quad \frac{G_a(X)(E) \vdash lis}{G_s(X = E) \vdash lis}
$$

$$
(\mathtt{mOp}) \quad \frac{\begin{array}{cc} t_1 \ \mathtt{is\ a\ fresh\ var} & G_a(t_1)(E_1) \vdash lis_1 \\ t_2 \ \mathtt{is\ a\ fresh\ var} & G_a(t_2)(E_2) \vdash lis_2 \\ \multicolumn{2}{c}{l \ \mathtt{is\ a\ fresh\ label}} \end{array}}{G_a(X)(E_1 OP E_2) \vdash lis_1 + lis_2 + [l : X \leftarrow t_1 OP t_2]}
$$

$$
(\mathtt{mParen}) \quad \frac{G_a(X)(E) \vdash lis}{G_a(X)((E)) \vdash lis}
$$

From SIMP to PA - Maximal Munch

Given `z = x - (y + 1)`, we have

```
Ga(z)(x - (y + 1)) ---> #(mOp)
    Ga(t1)(x) ---> #(mVar)
    [3:t1 <- x]

    Ga(t2)((y+1)) ---> #(mParen)
    Ga(t2)(y+1) ---> #(mOp)
        Ga(t3)(y) ---> #(mVar)
        [4:t3 <- y]
        Ga(t4)(1) ---> #(mConst)
        [5:t4 <- 1]

    [4:t3 <- y, 5:t4 <- 1, 6:t2 <- t3 + t4]

[3:t1 <- x, 4:t3 <- y, 5:t4 <- 1, 6:t2 <- t3 + t4, 7:z <- t1 - t2]
```

labels 3,4,5,6,7 are fresh.

From SIMP to PA - Maximal Munch

To convert an `if` statement

$$
(\texttt{mIf}) \quad \frac{\begin{array}{c} t \texttt{ is a fresh var} \\ G_a(t)(E) \vdash lis_0 \\ l_{IfCondJ} \texttt{ is a fresh label} \\ G_s(S_2) \vdash lis_2 \\ l_{EndThen} \texttt{ is a fresh label} \\ l_{Else} \texttt{ is the next label(w/o incr)} \\ G_s(S_3) \vdash lis_3 \\ l_{EndElse} \texttt{ is a fresh label} \\ l_{EndIf} \texttt{ is the next label (w/o incr)} \\ lis_1 = [l_{IfCondJ} : ifn\ t\ goto\ l_{Else}] \\ lis_2' = lis_2 + [l_{EndThen} : goto\ l_{EndIf}] \\ lis_3' = lis_3 + [l_{EndElse} : goto\ l_{EndIf}] \end{array}}{G_s(if\ \ E\ \{S_2\}\ else\ \{S_3\}) \vdash lis_0 + lis_1 + lis_2' + lis_3'}
$$

From SIMP to PA - Maximal Munch

To convert an if statement

```
Gs(if z < 1 { x = x + 1 } else { x = 0 }) ---> # (mIf)
    Ga(t6)(z<1) --->* #(mOp, mVar, mConst)
    [8:t7 <- z, 9:t8 <- 1, 10:t6 <- t7 < t8]
    lifCondJ = 11
    Gs(x = x + 1) --->* #(mAssign, mOp, mVar, mConst)
    [12:t9 <- x, 13:t10 <- 1, 14: x <- t9 + t10]
    lendThen = 15
    lElse = 16
    Gs(x = 0) --->* #(mAssign, mConst)
    [16:x <- 0]
    lendElse = 17
    lendIf= 18

[8:t7 <- z, 9:t8 <- 1, 10:t6 <- t7 < t8, #lis0
11:ifn t6 goto 16, # lis1
12:t9 <- x, 13:t10 <- 1, 14: x <- t9 + t10, 15:goto 18, # lis2'
16:x <- 0, 17:goto 18 # lis3'
]
```

Wait, isn't `17:goto 18` redundant?

From SIMP to PA - Maximal Munch

What if?

$$
(\texttt{mIf}') \quad \frac{\begin{array}{c} t \texttt{ is a fresh var} \\ G_a(t)(E) \vdash lis_0 \\ l_{IfCondJ} \texttt{ is a fresh label} \\ G_s(S_2) \vdash lis_2 \\ l_{EndThen} \texttt{ is a fresh label} \\ l_{Else} \texttt{ is the next label(w/o incr)} \\ G_s(S_3) \vdash lis_3 \\ l_{EndIf} \texttt{ is the next label (w/o incr)} \\ lis_1 = [l_{IfCondJ} : ifn\ t\ goto\ l_{Else}] \\ lis_2' = lis_2 + [l_{EndThen} : goto\ l_{EndIf}] \end{array}}{G_s(if\ \ E\ \{S_2\}\ else\ \{S_3\}) \vdash lis_0 + lis_1 + lis_2' + lis_3}
$$

From SIMP to PA - Maximal Munch

```
Gs(if z < 1 { x = x + 1 } else { x = 0 }) ---> # (mIf')
    Ga(t6)(z<1) --->* #(mOp, mVar, mConst)
    [8:t7 <- z, 9:t8 <- 1, 10:t6 <- t7 < t8]
    lifCondJ = 11
    Gs(x = x + 1) --->* #(mAssign, mOp, mVar, mConst)
    [12:t9 <- x, 13:t10 <- 1, 14: x <- t9 + t10]
    lendThen = 15
    lElse = 16
    Gs(x = 0) --->* #(mAssign, mConst)
    [16:x <- 0]
    lendIf= 17

[8:t7 <- z, 9:t8 <- 1, 10:t6 <- t7 < t8, #lis0
11:ifn t6 goto 16, # lis1
12:t9 <- x, 13:t10 <- 1, 14: x <- t9 + t10, 15:goto 17, # lis2'
16:x <- 0 # lis3
]
```

The last goto is eliminated. For now we stick with (`mIf`), as it is clearer to identify the end of if for debugging purposes.

From SIMP to PA - Maximal Munch

While loop

$$
(\texttt{mWhile}) \quad \frac{\begin{array}{c} l_{While} \texttt{ is the next label (w/o incr)} \\ t \texttt{ is a fresh var} \\ G_a(t)(E) \vdash lis_0 \\ l_{WhileCondJ} \texttt{ is a fresh label} \\ G_s(S) \vdash lis_2 \\ l_{EndBody} \texttt{ is a fresh label} \\ l_{EndWhile} \texttt{ is the next label (w/o incr)} \\ lis_1 = [l_{WhileCondJ} : ifn\ t\ goto\ l_{EndWhile}] \\ lis_2' = lis_2 + [l_{EndBody} : goto\ l_{While}] \end{array}}{G_s(while\ E\ \{S\}) \vdash lis_0 + lis_1 + lis_2'}
$$

From SIMP to PA - Maximal Munch

Assuming the next available label is 19

```
Gs(while x < y { x = x + 1; }) ---> #(mWhile)
    lwhile = 19
    Ga(t11)(x < y) --->* #(mOp, mVar)
    [19:t12 <- x, 20:t13 <- y, 21: t11 <- t12 < t13]

    lwhilecondj = 22
    Gs(x = x + 1) --->* #(mAssign, mOp, mVar, mpConst)
    [23:t14 <- x, 24:t15 <- 1, 25: x <- t14 + t15]

    lendBody = 26
    lendWhile = 27
[19:t12 <- x, 20:t13 <- y, 21: t11 <- t12 < t13, # lis0
22:ifn t11 goto 27, # lis1
23:t14 <- x, 24:t15 <- 1, 25: x <- t14 + t15, 26: goto 19 # lis2'
]
```

From SIMP to PA - Maximal Munch

The rest of the rules

$$(\texttt{mReturn}) \quad \frac{G_a(r_{ret})(X) \vdash lis \quad l \texttt{ is a fresh label}}{G_s(return\ X) \vdash lis + [l : ret]}$$

$$(\texttt{mSequence}) \quad \frac{\texttt{for } l \in \{1, n\} \quad G_s(S_l) \vdash lis_l}{G_s(S_1; ...; S_n) \vdash lis_1 + ... + lis_n}$$

$$(\texttt{mNOp}) \quad G_s(nop) \vdash []$$

Issue with Maximal Munch

- Using too many temp variables.

```
Gs(z = x - (y + 1)) ---> # (mAssign)
Ga(z)(x - (y + 1)) --->
[3:t1 <- x, 4:t3 <- y, 5:t4 <- 1, 6:t2 <- t3 + t4, 7:z <- t1 - t2]
```

But we could encode it using

```
[3: t1 <- y + 1, 4:t2 <- x - t1, 5:z <- t2]
```

Maxmimal Munch V2

Replacing all the use of $G_a(X)(E) \vdash lis$ by $G_e(E) \vdash (\hat{e}, \check{e})$, where

- $\check{e}$ is a sequence of label instructions generated from E and
- $\hat{e}$ is the "result" operand storing the final result of $\check{e}$.

$$(\texttt{m2Assign}) \quad \frac{G_e(E) \vdash (\hat{e}, \check{e}) \quad l \texttt{ is a fresh label.}}{G_s(X = E) \vdash \check{e} + [l : X \leftarrow \hat{e}]}$$

Maxmimal Munch V2

Replacing all the use of $G_a(X)(E) \vdash lis$ by $G_e(E) \vdash (\hat{e}, \check{e})$, where

- $\check{e}$ is a sequence of label instructions generated from E and
- $\hat{e}$ is the "result" operand storing the final result of $\check{e}$.

$$(\texttt{m2Const}) \quad G_e(C) \vdash (conv(C), [])$$

$$(\texttt{m2Var}) \quad G_e(Y) \vdash (Y, [])$$

$$(\texttt{m2Paren}) \quad \frac{G_e(E) \vdash (\hat{e}, \check{e})}{G_e((E)) \vdash (\hat{e}, \check{e})}$$

$$(\texttt{m2Op}) \quad \frac{\begin{array}{c} G_e(E_1) \vdash (\hat{e}_1, \check{e}_1) \\ G_e(E_2) \vdash (\hat{e}_2, \check{e}_2) \\ t \texttt{ is a fresh variable.} \\ l \texttt{ is a fresh label.} \end{array}}{G_e(E_1 OP E_2) \vdash (t, \check{e}_1 + \check{e}_2 + [l : t \leftarrow \hat{e}_1 OP \hat{e}_2])}$$

Maxmimal Munch V2

```
Gs(z = x - (y + 1)) ---> #(m2Assign)
    Ge(x - (y + 1)) ---> #(m2Op)
        Ge(x) ---> #(m2Var)
        (x, [])
        Ge(y + 1) ---> #(m2Op)
            Ge(y) ---> #(m2Var)
            (y, [])
            Ge(1) ---> #(m2Const)
            (1, [])
        (t1, [3: t1 <- y + 1])
    (t2, [3: t1 <- y + 1, 4:t2 <- x - t1])
[3: t1 <- y + 1, 4:t2 <- x - t1, 5:z <- t2]
```

Can we further reduce it to

```
[3: t1 <- y + 1, 4:z <- x - t1]
```

It could work in this example but does not work in general.

Adjusting other rules

$$
(\texttt{m2If})\quad
\frac{
\begin{array}{c}
G_e(E) \vdash (\hat{e}, \check{e}) \\
l_{IfCondJ} \ \texttt{is a fresh label} \\
G_s(S_2) \vdash lis_2 \\
l_{EndThen} \ \texttt{is a fresh label} \\
l_{Else} \ \texttt{is the next label(w/o incr)} \\
G_s(S_3) \vdash lis_3 \\
l_{EndElse} \ \texttt{is a fresh label} \\
l_{EndIf} \ \texttt{is the next label (w/o incr)} \\
lis_1 = [l_{IfCondJ} : ifn\ \hat{e}\ goto\ l_{Else}] \\
lis_2' = lis_2 + [l_{EndThen} : goto\ l_{EndIf}] \\
lis_3' = lis_3 + [l_{EndElse} : goto\ l_{EndIf}]
\end{array}
}{
G_s(if\ \ E\ \{S_1\}\ else\ \{S_2\}) \vdash \check{e} + lis_1 + lis_2' + lis_3'
}
$$

Adjusting other rules

$$(\texttt{m2Return}) \qquad G_s(return\ X) \vdash \check{e} + [l_1 : r_{ret} \leftarrow X, l_2 : ret]$$

$$(\texttt{m2While}) \qquad \frac{\begin{array}{c} l_{While} \texttt{ is the next label (w/o incr)} \\ G_e(E) \vdash (\hat{e}, \check{e}) \\ l_{WhileCondJ} \texttt{ is a fresh label} \\ G_s(S) \vdash lis_2 \\ l_{EndBody} \texttt{ is a fresh label} \\ l_{EndWhile} \texttt{ is the next label (w/o incr)} \\ lis_1 = [l_{WhileCondJ} : ifn\ \hat{e}\ goto\ l_{EndWhile}] \\ lis_2' = lis_2 + [l_{EndBody} : goto\ l_{While}] \end{array}}{G_s(while\ E\ \{S\}) \vdash \check{e} + lis_1 + lis_2'}$$

Summary

- SIMP program
- Pseudo Assembly (3 address IR)
- Maximal Munch Algorithms