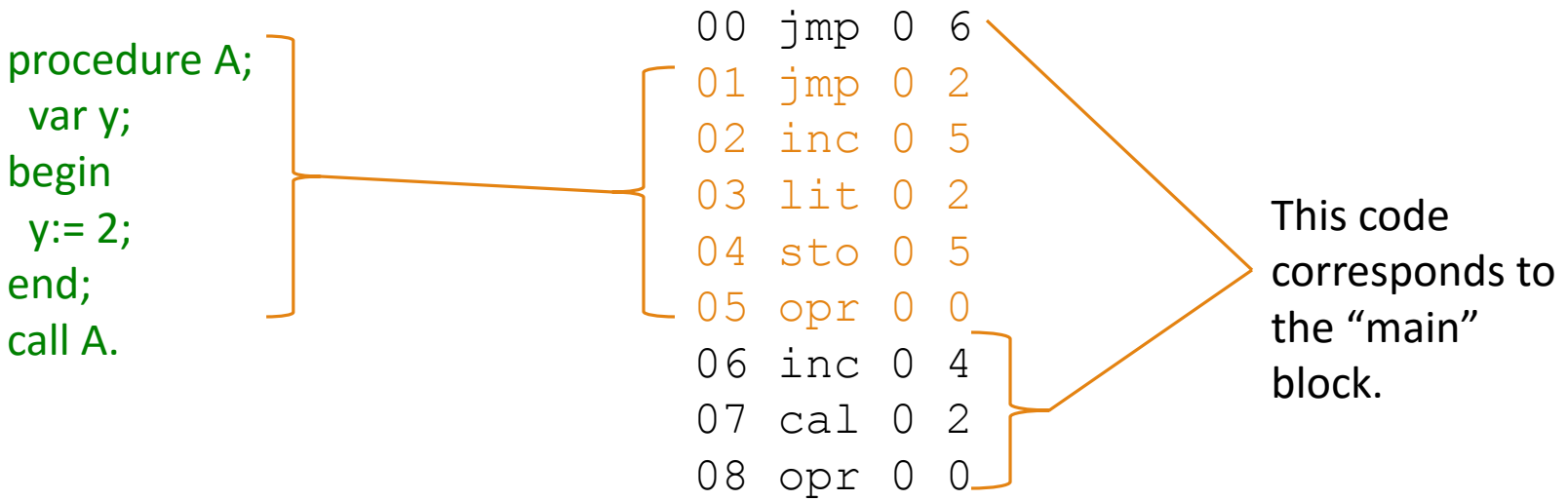
Recitation 7: Procedure Code Generation 1

COP3402 FALL 2015 – ARYA POURTABATABAIE

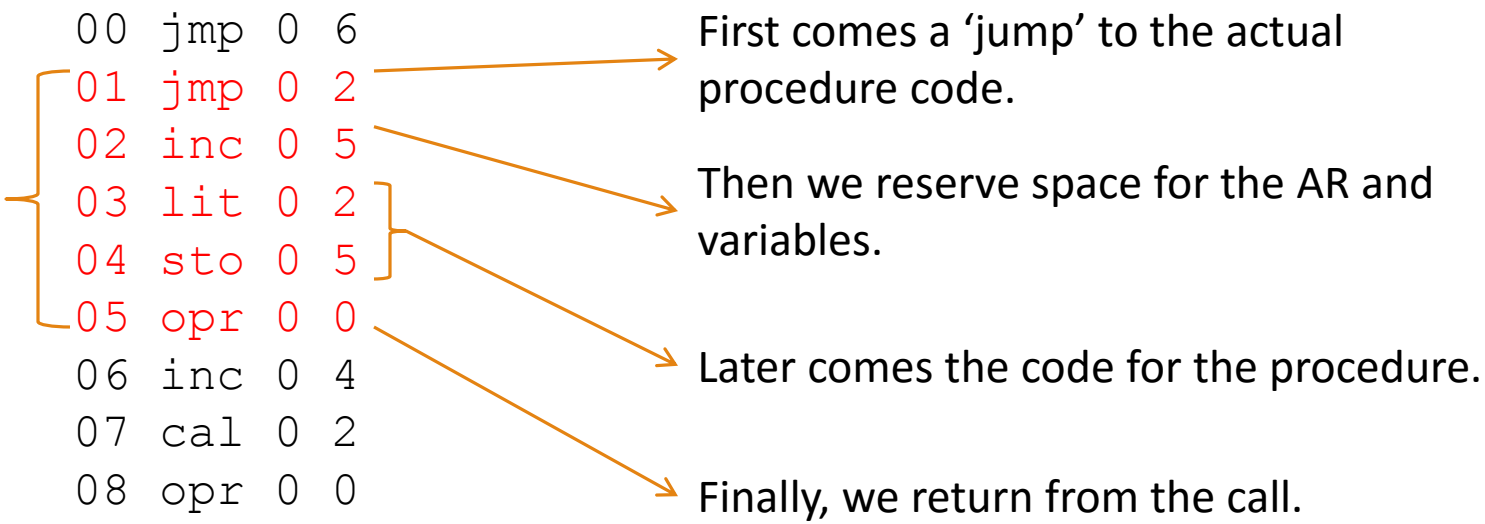
FROM EURIPIDES MONTAGNE, FALL 2014

Procedures: PM/0 code look

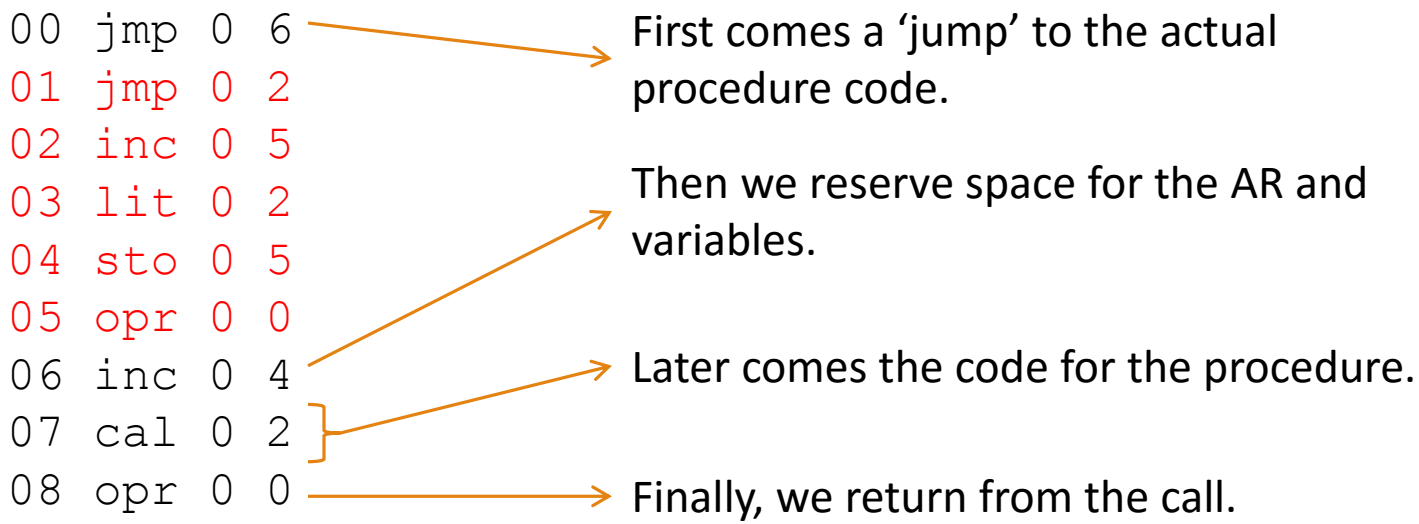
In general, when a procedure is compiled, the machine code will look like this:



Procedures: PM/0 code look



Procedures: PM/0 code look



How do I know the address for the first jump?

Step 1: The “jump” operation

jmpCodeAddr = 00

When we have nested procedures, we don’t know where to jump at the time we have to generate the jump operation.

```
00 jmp 0 ??
```

We must remember where the jump is in order to replace the correct address later.

Step 1: The “jump” operation

jmpCodeAddr = 00

When we have nested procedures, we don’t know where to jump at the time we have to generate the jump operation.

We must remember where the jump is in order to replace the correct address later.

```
00 jmp 0 ??
01 jmp 0 2
02 inc 0 5
03 lit 0 2
04 sto 0 5
05 opr 0 0
```

Step 1: The “jump” operation

jmpCodeAddr = 00

When we have nested procedures, we don’t know where to jump at the time we have to generate the jump operation.

We must remember where the jump is in order to replace the correct address later.

```
00 jmp 0 ??
01 jmp 0 2
02 inc 0 5
03 lit 0 2
04 sto 0 5
05 opr 0 0
06 inc 0 4
…
```

Step 1: The “jump” operation

jmpCodeAddr = 00

When we have nested procedures, we don’t know where to jump at the time we have to generate the jump operation.

We must remember where the jump is in order to replace the correct address later.

```
00 jmp 0 6
01 jmp 0 2
02 inc 0 5
03 lit 0 2
04 sto 0 5
05 opr 0 0
06 inc 0 4
…
```

Step 1: The “jump” operation

When we parse the procedure declaration, we inserted it’s name into the Symbol Table, and the address is the next code address, which is the address of the initial JMP.

Once we know the address where the procedure code starts, we must:

- Update the jump operation.

Step 2: Reserve space

A procedure must reserve space for it's activation record and variables, before its actual code starts.

We must keep track of how much space we should reserve. It should be 4 at least (for the AR), and must increase for each declared variable.

Be careful: each procedure must keep track of their own space, don't use a global variable.

To reserve space, we generate a "inc" operation.

- gen(INC, 0, SPACE);

Step 3: Procedure code generation

Now we can generate code for the procedure.

This will be handle by the “statement” function.

Step 4: The return call

This one is simple: just generate a return operation (opr 0 0).

Code Generation for Procedure

Step 1: Generate a jump operation.

- Store the address of this jump.
- Keep parsing/generating code.
- Update the jump address.

Step 2: Reserve Space

Step 3: Generate the procedure code (parse the statements).

Step 4: Generate a return operation.

Who will handle all this?

The code generation for a procedure is handle by the block function.

- <proc-decl> ::= procedure <ident> ; <block> ; | e
- <program> ::= <block> .
- <block> ::= <const-decl> <var-decl> <proc-decl> <statement>

<block> Parser Procedure

<block> ::= <const-decl> <var-decl> <proc-decl> <statement>

```
procedure BLOCK();
begin
 if TOKEN = “const” then CONST-DECL();
 if TOKEN = “var” then VAR-DECL();
 if TOKEN = “procedure” then  PROC-DECL();
 STATEMENT();
end;
```

<block> Parser and Code Generation

```
procedure BLOCK();
begin
 space = 4;
 jmpaddr = gen(JMP, 0, 0);  // Step 1
 if TOKEN = “const” then CONST-DECL();
 if TOKEN = “var” then space += VAR-DECL();
 if TOKEN = “procedure” then  PROC-DECL();
 code[jmpaddr].addr = NEXT_CODE_ADDR;
 gen(INC, 0, space); // Step 2
 STATEMENT(); // Step 3
 gen(OPR, 0, 0); // Step 4
end;
```

Keep track of lexicographical level

We need to set the correct lexicographical each time we generate a LOD, STO or CAL operation.

One way is to keep track of the current lexicographical level in a global variable.

The lexicographical level increases when we enter the BLOCK procedure, and decreases when we exit it.

<block> Parser and Code Generation

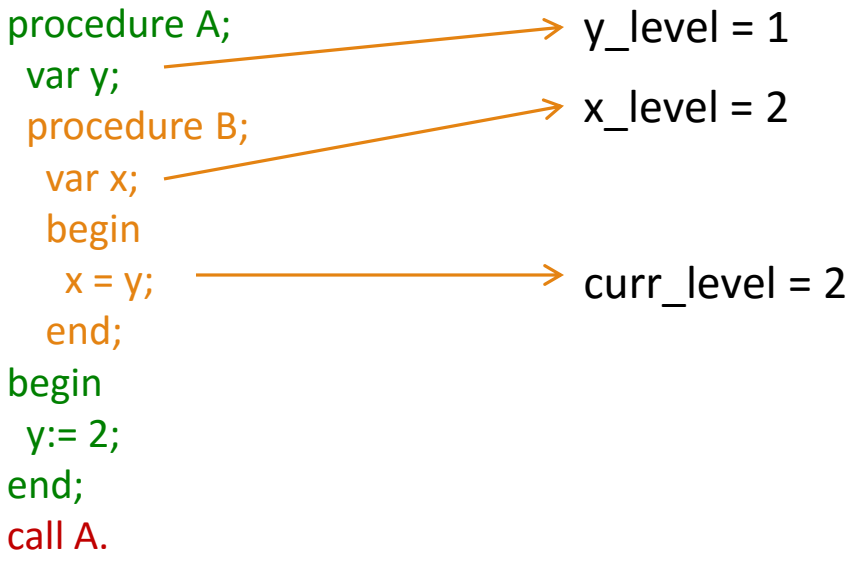
```
procedure BLOCK();
begin
  level++;
  space = 4;
  jmpaddr = gen(JMP, 0, 666);
  if TOKEN = “const” then CONST-DECL();
  if TOKEN = “var” then space += VAR-DECL();
  if TOKEN = “procedure” then  PROC-DECL();
  code[jmpaddr].addr = NEXT_CODE_ADDR;
  gen(INC, 0, space);
  STATEMENT();
  gen(RTN, 0, 0);
  level--;
end;
```

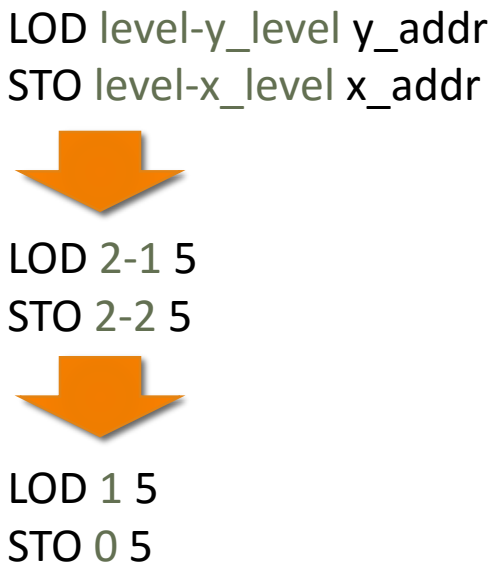
Keep track of lexicographical level

To generate a LOD or STO, the correct lexicographical level is given by:

level – var_level

For example:





call Code Generation

At this point, we already know the procedure address, so the call code is easy to generate.

```
 procedure STATEMENT;
begin
      …
      else if TOKEN = "call" then begin
        GET_TOKEN();
        if TOKEN <> IDENT then ERROR (missing identifier);
        GET_TOKEN();
      end
…
```

call Code Generation

At this point, we already know the procedure address, so the call code is easy to generate.

```
 procedure STATEMENT;
begin
      …
      else if TOKEN = "call" then begin
        GET_TOKEN();
        if TOKEN <> IDENT then ERROR (missing identifier);
        i = find(TOKEN);
        if i == 0 then ERROR (Undeclared identifier);
         if symboltype(i) == PROCEDURE then gen(CAL, level-symbollevel(i), symboladdr(i));
        else ERROR(call must be followed by a procedure identifier);
        GET_TOKEN();
      end
…
```

Questions?