COMP310: Legacy Game Systems

4: Further NES Assembly

Writing maintainable code

Using multiple files

```
.include "MyFile.asm"
```

Named variables

```
.rsset $0000     ;put variables starting at 0
score1   .rs 1   ;put score for player 1 at $0000
score2   .rs 1   ;put score for player 2 at $0001
buttons1 .rs 1   ;put controller data for p1 at $0002
buttons2 .rs 1   ;put controller data for p2 at $0003
```

Named constants

```
RIGHTWALL       = $02
TOPWALL         = $20
BOTTOMWALL      = $D8
LEFTWALL        = $F6
```

Local labels

```
ReadA:
   ...
   BEQ ReadADone
LoopA:
   ...
   CPX #$10
   BNE LoopA
ReadADone:

ReadB:
   ...
   BEQ ReadBDone
LoopB:
   ...
   CPX #$10
   BNE LoopB
ReadBDone:
```

=

```
ReadA:
   ...
   BEQ .Done
.Loop:
   ...
   CPX #$10
   BNE .Loop
.Done:

ReadB:
   ...
   BEQ .Done
.Loop:
   ...
   CPX #$10
   BNE .Loop
.Done:
```

Local labels

- Name begins with .
- Associated with the **preceding** global (i.e. non-local) label

First draft — why doesn't this work?

```
ReadA:
   ...
   BEQ .Done
LoopA:
   ...
   CPX #$10
   BNE LoopA
.Done:

ReadB:
   ...
   BEQ .Done
LoopB:
   ...
   CPX #$10
   BNE LoopB
.Done:
```

Subroutines

```
Add4ToX:
   INX
   INX
   INX
   INX
   RTS
```

```
   ; Usage:
   JSR Add4ToX
```

Macros

```
Add4ToX .macro
   INX
   INX
   INX
   INX
   .endm
```

```
   ; Usage:
   Add4ToX
```

Subroutines vs macros

- Macros are expanded at **assembly time**
- Subroutines are called at **run time**
- Macros have less **CPU overhead**, at the expense of **program size**
- Macros can take **arguments**

Fun with macros

How does this work?

```
AddToX .macro
   .if \1 > 0
     INX
     AddToX \1-1
   .endif
   .endm
```

```
   ; Usage:
   AddToX 4
```

More tips

Read the `usage.txt` file from `NESASM3.zip`

Workshop