6502 DEBUGGER

Haley Whitman
Alison Legge
Frazer Bayley
Jeremy Brennan
Abdualziz AL-Heidous

CIS 422 – Dr. Faulk

BACKGROUND

Project is split into two deliverable parts

1. MOS Technology 6502 Processor
2. Debugging Interface for 6502 Assembly

Requirements for 6502 Processor

1. Accurately emulate the behavior of a 6502 processor.
2. Allow for third-party control of the processor.

Requirements for the debugging interface

1. Allow for a user to input 6502 assembly
2. Allow for the user to view the results, and step through their inputted program.

PROJECT PLAN

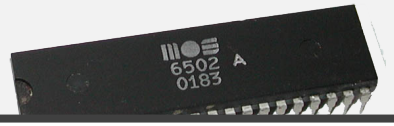
Followed an iterative design, split between

1. Creating the processor.
2. Creating the debugging interface.

We created the module design for the processor from Synertek and MOS Technology's 6502 Programming/Hardware manual.

We followed customer requests and typical debugging use-cases to design the debugging interface. Our documentation focused much more heavily on the debugging interface.

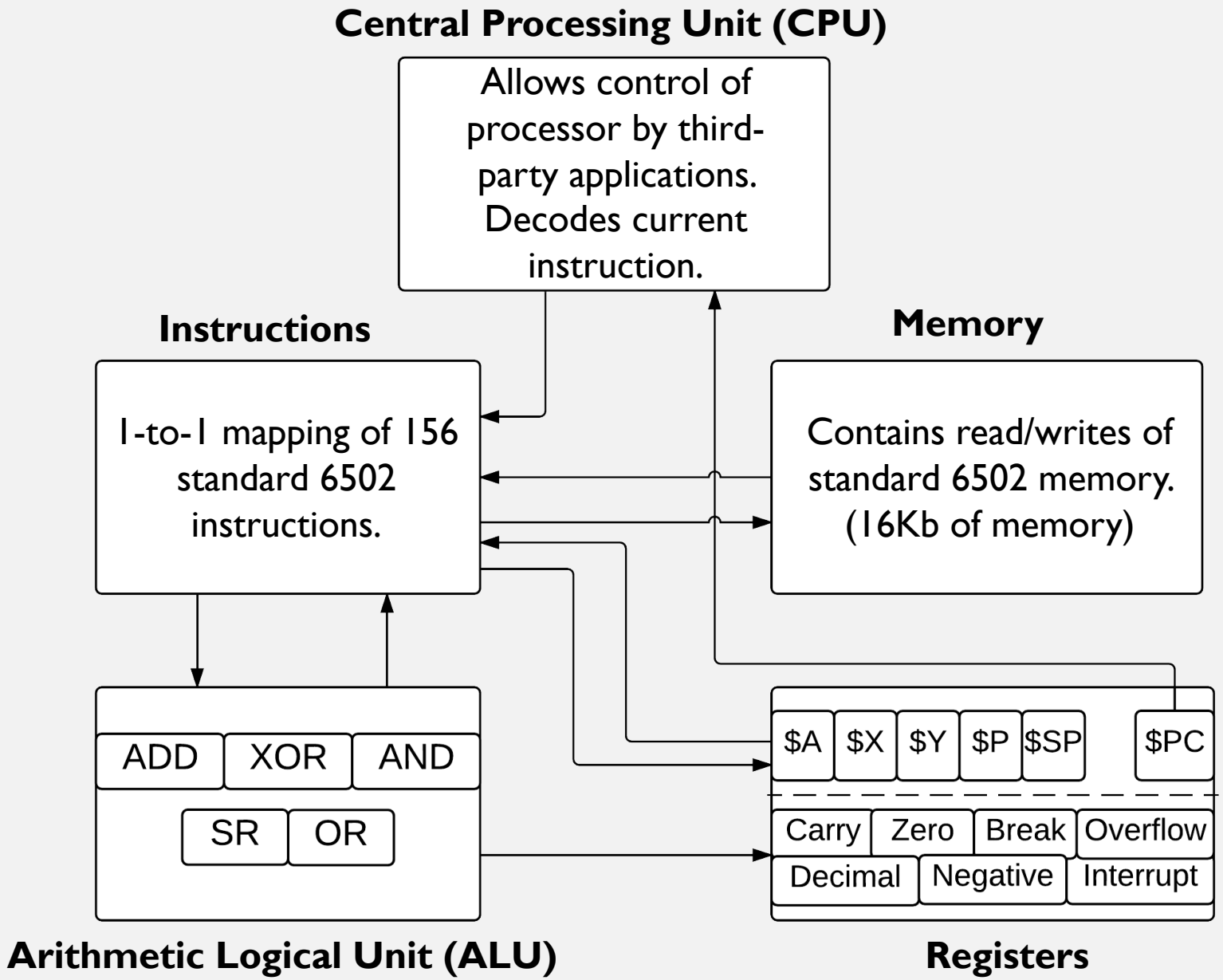
We have met all the customer requirements laid out for us, and documentation needed for the class. Deliverables are laid out in the following slides.



PROCESSOR DESIGN

Fairly typical design specification

Follows original dataflow of 6502 processor

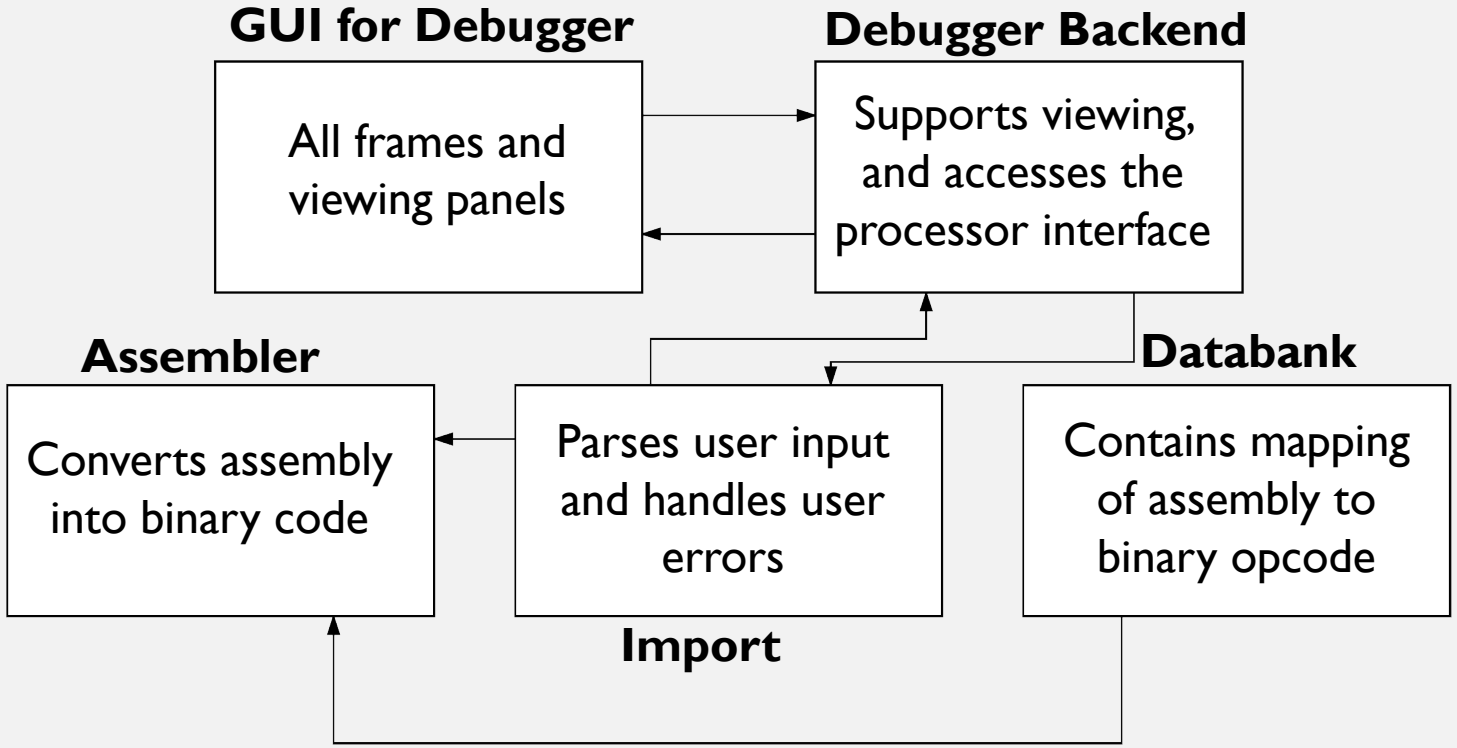


DEBUGGER DESIGN

Fairly typical debugging interface, modeled after more retro-based debuggers for processors such as the Gameboy, where analyzing memory output is helpful.

Has to easily allow for further features.

Written completely separate from the processor



CUSTOMER REQUESTS

Three Main Points

1. Wanted to be able to load any set of standard 6502 programs.
2. Be able to fully execute, or step through the program in a variety of ways.
3. Be able to easily view the processor changing line by line

Customer Iterations

1. We continually presented the customer with the product, and received feedback on what to alter.
2. Multiple behavioral changes, involving controlling the flow of the processor.
3. Few interface changes.

QA PLAN

Code Reviews

1. All processor, and specific debugger module code have unit tests written for them. This allowed for regression testing.
2. Overall code behavior was done by running a suite of 6502 assembly programs. Processor behavior was compared with other 6502 emulators.

Documentation Reviews

1. Feedback was received for all documentation, and reviews were completed by two people per document, each focusing on specific aspects of the document.

LESSON'S LEARNED

Documentation

1. Documentation drafts were mostly on schedule. There were some issues with getting reviews implemented into the main documentation. This was addressed by doing more complete reviews after the initial review.
2. Needed a better way of organizing requests made for the documentation in the classroom.

Coding

1. We had a great module structure, allowed for a lot of concurrent work, and was very effective for our team.
2. We employed heavy use of pair-programming, which greatly helped in understanding the flow of the processor.

COURSE IMPROVEMENTS

Suggestions

1. Groups complete one project instead of two.
    - All groups work concurrently on the same portion of the project.
    - Documentation is completed and reviewed before the code is started.
    - Increases everyone's understanding of every part of software development.

2. Customer always be the Professor.
    - This way the requirements and expectations are always clear.

APPLICATION

6502 Emulator & Debugger

File | Assemble | Execute | Step | Step To | Memory Dump

```
LDX #$01; x = 1
STX $00; stores x
SEC; clean carry;
LDY #$07; calculates 7th fibonacci number
TYA; transfer y register to accumulator
SBC #$03; handles the algorithm iteration counting
TAY; transfer the accumulator to the y register
CLC; clean carry
LDA #$02; a = 2
STA $01; stores a
loop: LDX $01; x = a
      ADC $00; a += x
      STA $01; stores a
      STX $00; stores x
      DEY; y -= 1
```

```
HRAM:1F8 0000
HRAM:1FA 0000
HRAM:1FC 0000
HRAM:1FE 0000
$200: A2 01    LDX #$01
$202: 86 00    STX $00
$204: 38       SEC
$205: A0 07    LDY #$07
$207: 98       TYA
$208: E9 03    SBC #$03
$20A: A8       TAY
$20B: 18       CLC
$20C: A9 02    LDA #$02
$20E: 85 01    STA $01
$210: A6 01    LDX $01
$212: 65 00    ADC $00
$214: 85 01    STA $01
$216: 86 00    STX $00
$218: 88       DEY
```

```
$0200:  A2 01 86 00 38 A0 07 98 | E9 03 A8 18 A9 02 85 01 | ∘□□.8 □□é□¨□⊘□□□
$0210:  A6 01 65 00 85 01 86 00 | 88 D0 10 00 00 00 00 00 | ¦□e.□□□.□Ð□.....
$0220:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | ................
$0230:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | ................
$0240:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | ................
$0250:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | ................
$0260:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | ................
$0270:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | ................
$0280:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | ................
$0290:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | ................
```

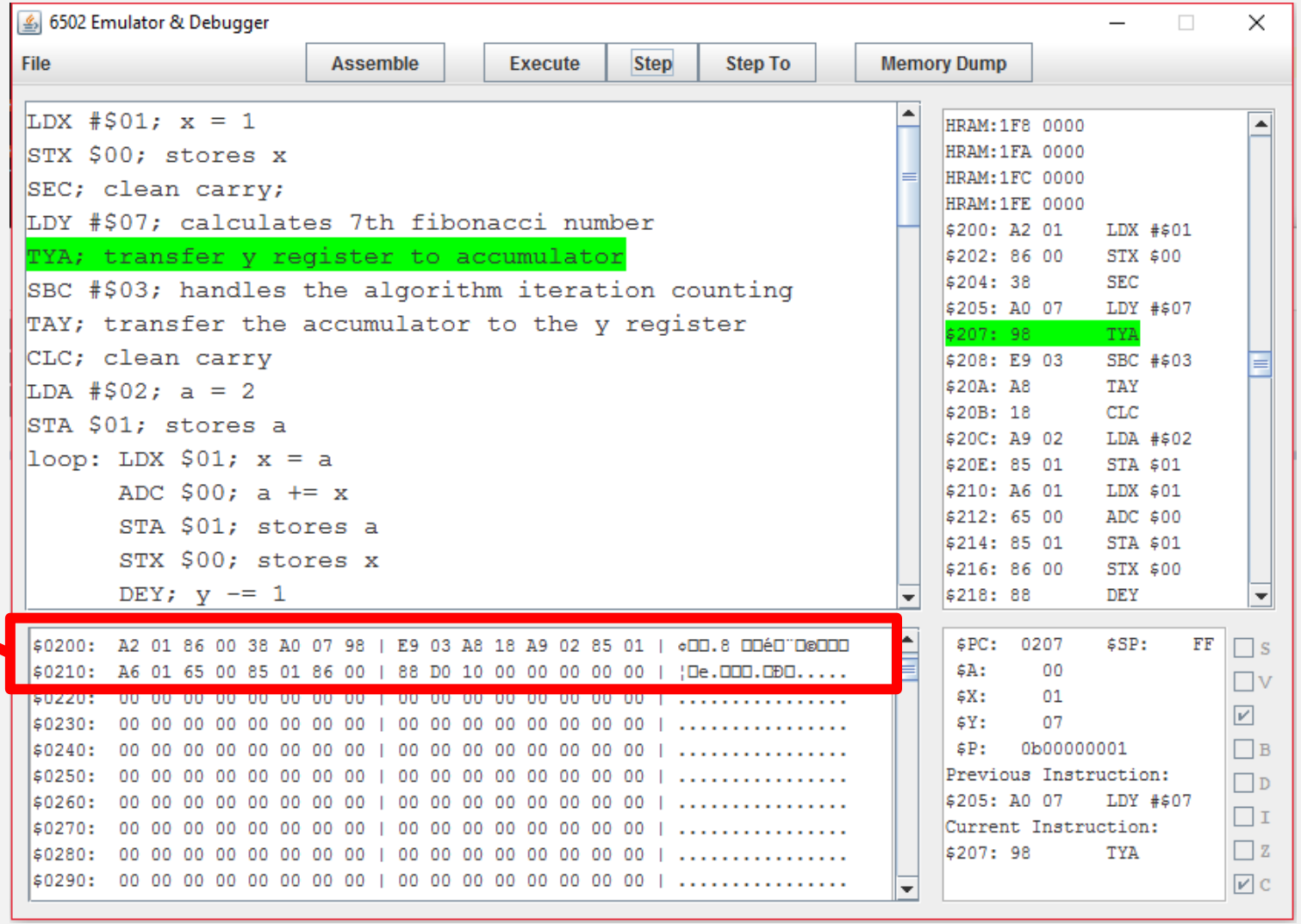
```
$PC:   0207     $SP:    FF
$A:      00
$X:      01
$Y:      07
$P:    0b00000001
Previous Instruction:
$205: A0 07    LDY #$07
Current Instruction:
$207: 98       TYA
```

☐ S ☐ V ☑ ☐ B ☐ D ☐ I ☐ Z ☑ C

APPLICATION

Memory gets stored, and updated with every CPU cycle. This is easily viewed in base-16, split every 16 bytes.

The ASCII memory representation is shown on the left.

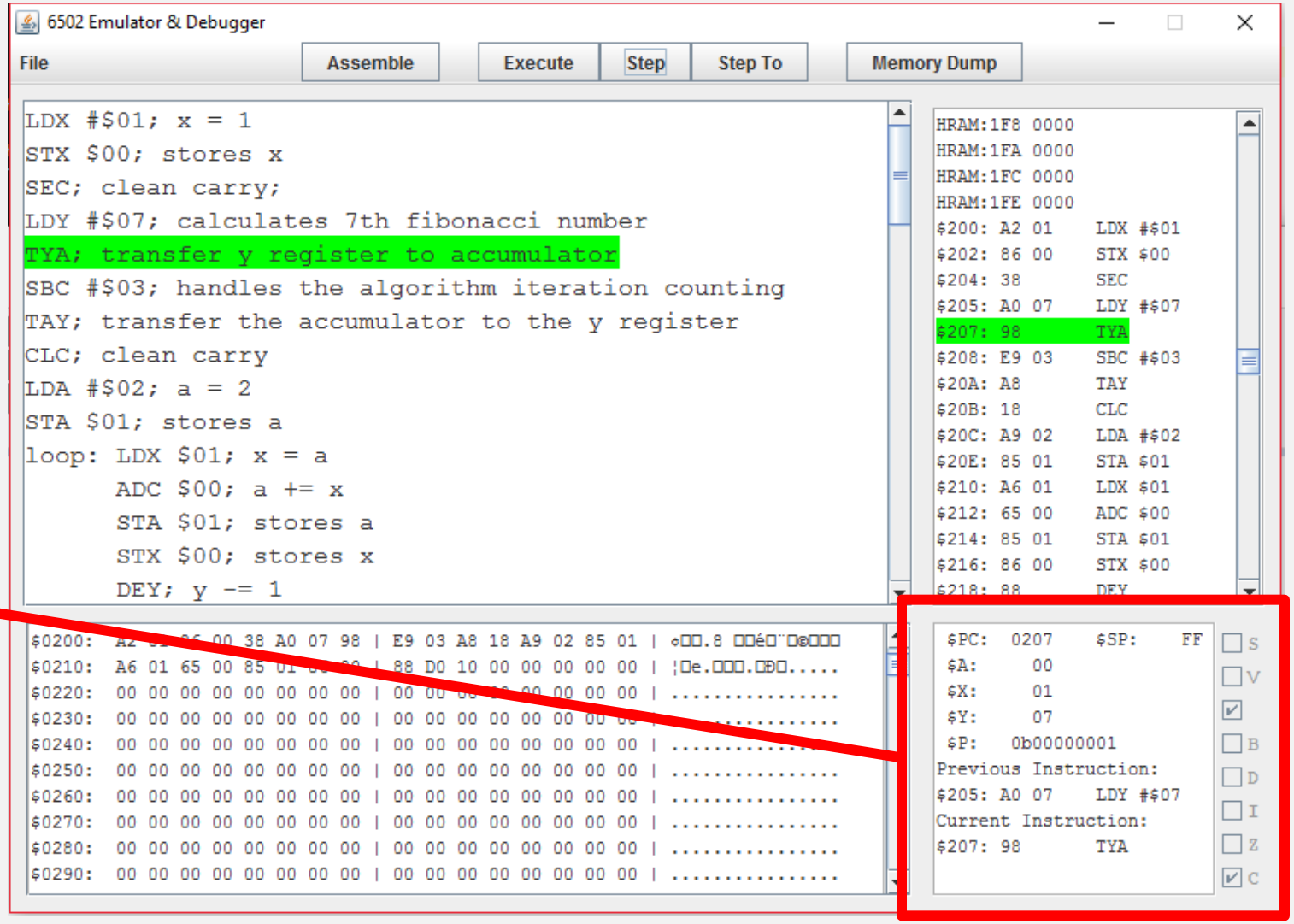


APPLICATION

All register values are updated at every cycle (Shown in base-16).

The previous and currently executed instruction are also shown

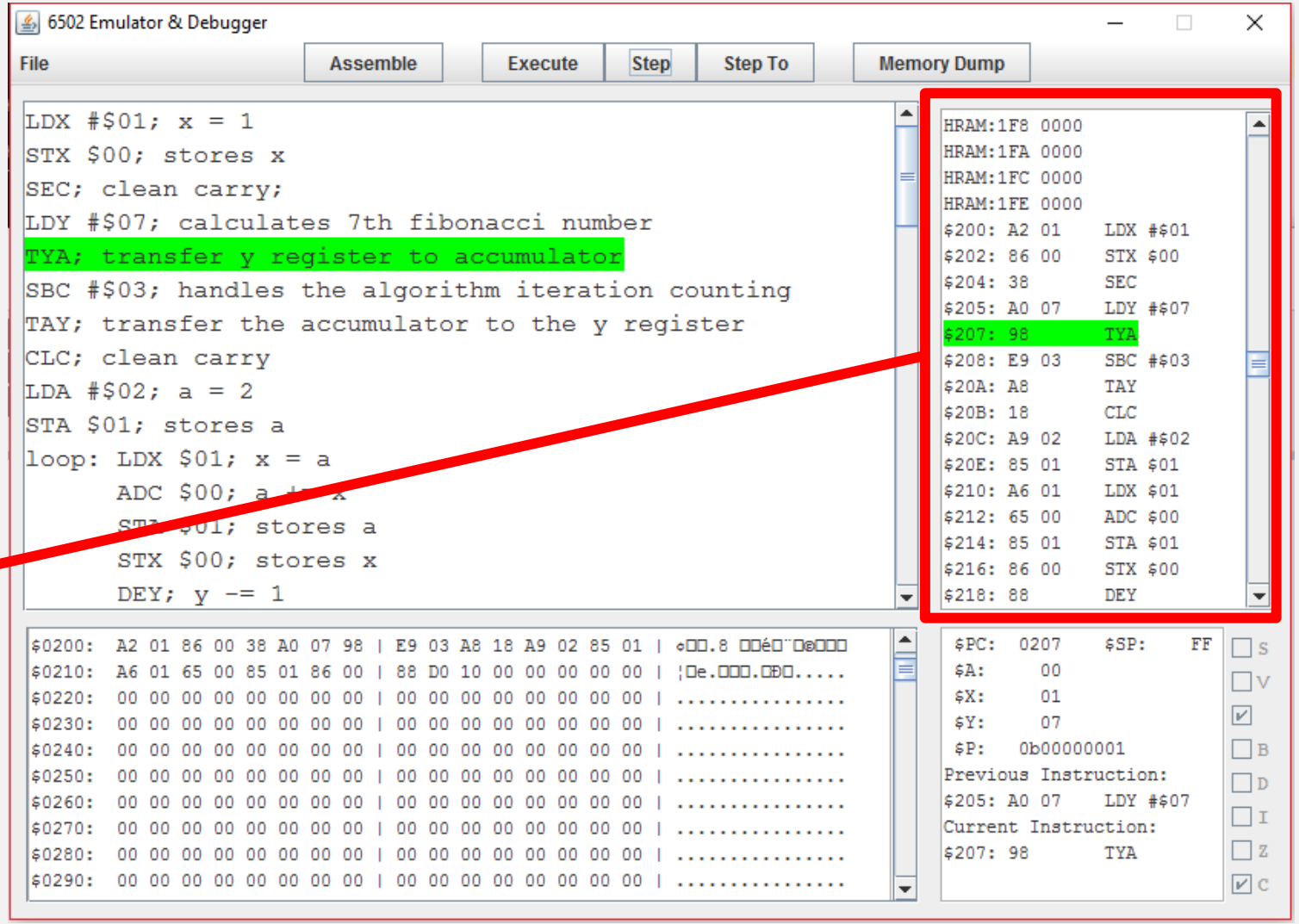
Register flags are visualized with checkboxes, if they are currently set.



APPLICATION

A more detailed look at the processor's Zero page, High-Index Ram, and starting ROM.

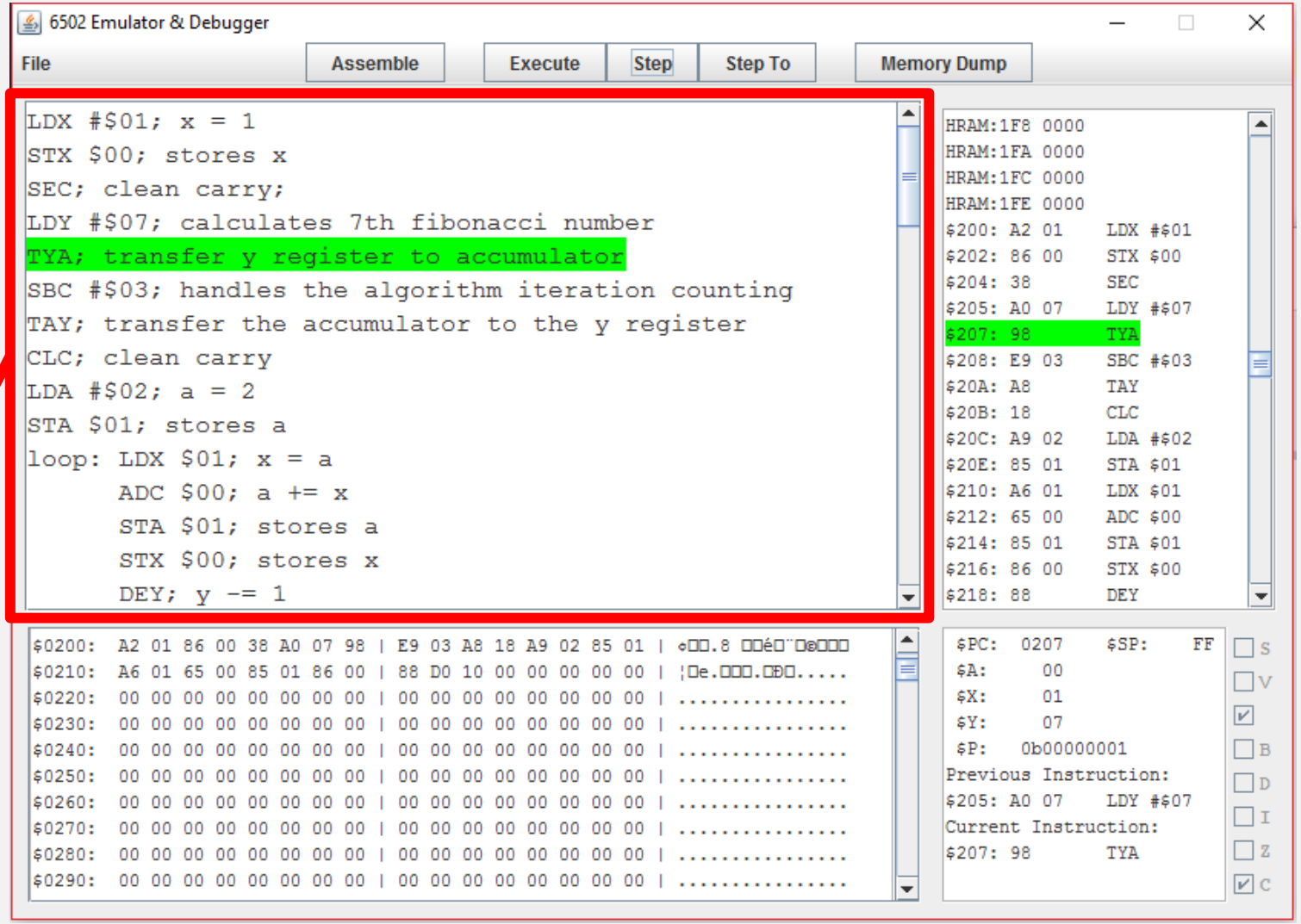
Opcodes and values are displayed for instructions, and their assembly translation is also shown and highlighted at each step.



APPLICATION

The user can type, import, or copy and paste standard 6502 assembly into this window.

Once assembled, the currently executed line will be highlighted as the program gets ran.



APPLICATION

All programs must be assembled, before any execution commands can be given.

Execute will fully run the program.

Step will run one cycle of the processor.

Step to will allow for a label to be chosen, and the processor will execute to there.