Term Project Alternative

CS 154: Formal Languages and Computability
Spring 2017

San José State University
Department of Computer Science

Ahmad Yazdankhah
ahmad.yazdankhah@sjsu.edu

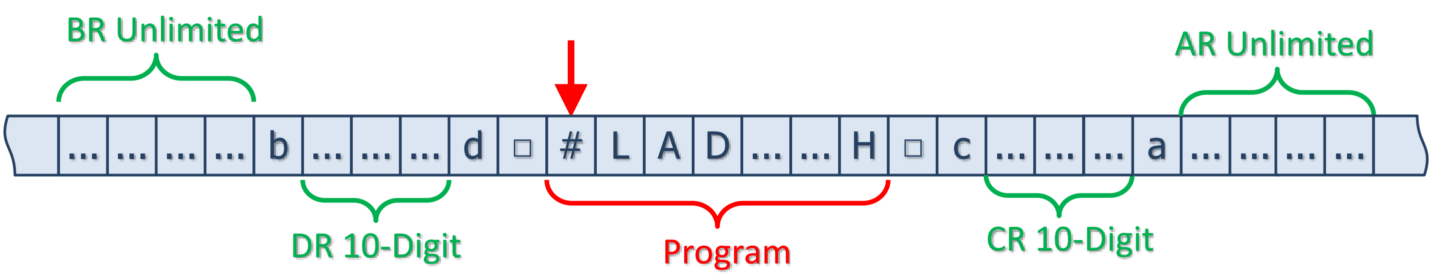
Objective

To **simulate** a simple microprocessor called **AYS17** by a Standard **Turing machine** with extra features of JFLAP.

AYS17 Structure

AYS17 has four registers called A, B, C, and D. A-register (AR for short) is called "**accumulator**" because the results of all arithmetic operations go to AR.

These four registers can be simulated on the tape of the TM as is shown in the following figure.



We use letters a, b, c and d at the beginning of each register to specify their territories. AR and BR are unlimited but CR and DR are 10-digit registers.

The content of AR is the **output** of the programs. Therefore, it should be implemented in such a way that at the end of the program, its content be shown appropriately.

At startup, your machine initializes these registers as the above figure shows and moves the head to the first instruction of the program. AYS17 runs the program from that point. The format of the programs, that should be encoded in one string, will be explained in the next sections.

Numbers

- All **numerical values** are prefixed with '$' and are represented in "unary notation".
- Negative numbers are prefixed with '$-'.
- There is NO limitation for the magnitude of the numbers!
- The only limitation is the contents of registers C and D that are 10 digits and the programmers (not you as the implementer) should make sure that they don't overflow.

Numbers Examples

AYS17 Representation	**Decimal Equivalent**
$	**zero**
$11	2
$11111	5
$-1	-1
$-111	-3
$-	error
11	error
$1A11	error
-$11	error

Assembly Language of AYS17

AYS17 has **nine distinct instructions** as the following table shows.

The values of X and Y might be: X ∈ {A, B, C, D}; Y ∈ {A, B, C, D, $NUM, NUM}

Mnemonic	Meaning	Description
LD X, Y	Load	Loads X register with Y. Y can be another register or a numerical value.
INC X	Increment	Increments the content of the register X by one
DEC X	Decrement	Decrements the content of the register X by one
ADD X	Add	Adds the operand X with the content of AR
SUB X	Subtract	Subtracts the operand X from the content of AR
JNZ X, L	Jump if not zero	Jumps to the label L if the content of the register X is **not zero** (can be positive or negative numbers)
JNG X, L	Jump if negative	Jumps to the label L if the content of the register X is **negative**
JMP L	Jump	Jumps unconditionally to the label L
HLT	Halt	Halts and prepares the appropriate outputs on AR and BR. All programs **should end** with HLT.

Note that we use **decimal numbers** prefixed with '$' (e.g. $13) when we write **assembly program** but we translate them into unary notation when we assemble the program to create object code.

For an example, refer to "Programming by AYS17" section.

Instructions Syntax

Syntax: **[label] operation operand1, operand2**

Syntax: **[label] operation operand1**

Syntax: **[label] operation**

The instructions of AYS17 have zero, one, or two operands and can be **optionally** preceded with a **unique** numerical label.

A label is **required** if the instruction is the destination of a jump instruction. If the required label is not provided, or if it is not unique or if it is not valid, then the behavior of the machine won't be predictable. Therefore, it is **AYS17 programmers'** responsibility to check the existence, validity, and uniqueness of the labels, **not you** as the implementer.

In general, an instruction may have 4 parts:

- **label**: is a **unique** positive, non-zero unary number without '$'
- **operation**: is an operation from the table mentioned in "assembly language of AYS17" section.
- **operand**: can be registers' letters (i.e. A, B, C, D), a numerical value, or a label.
  If it is a numerical value, we prefix it with '$' as described in "Numbers" section.
  If it is a label, for jump instructions, it does not need '$'.
  We represent numerical values with decimal notation prefixed with '$' when we write assembly programs but labels don't need to be prefixed with '$'.

Instructions Examples

Instruction	Description
3 HLT	Halts and shows the content of AR and BR
8 ADD $-2	Adds the negative number -2 with the content of AR
SUB $3	Subtracts the number 3 from the content of AR
5 JMP 8	Jumps to the instruction that is labeled by 2.

Assembly Codes of Instructions

In the following tables, the values of X and Y might be:

X ∈ {A, B, C, D}; Y ∈ {A, B, C, D, $NUM, NUM}

Load Operations

	A	**B**	**C**	**D**	**V**alue
LD A, Y	-	LAB	LAC	LAD	LAV
LD B, Y	LBA	-	LBC	LBD	LBV
LD C, Y	LCA	LCB	-	LCD	LCV
LD D, Y	LDA	LDB	LDC	-	LDV

Arithmetic Operations

	A	**B**	**C**	**D**	**V**alue
INC X	IA	IB	IC	ID	-
DEC X	DA	DB	DC	DD	-
ADD Y	-	AB	AC	AD	AV
SUB Y	-	SB	SC	SD	SV

Control Operations

	A	**B**	**C**	**D**	**L**abel
JNZ X, L	ZA	ZB	ZC	ZD	
JNG X, L	NA	NB	NC	ND	
JMP L	-	-	-	-	J
HLT	-	-	-	-	H

AYS17 has **9 distinct instructions** that can be written in **42 different forms**.

Programming by AYS17

We write the programs in assembly language. Then we encode the assembly program by the assembly codes provided in "assembly codes of instructions" section.

We input the encoded program, that is one string, on the tape of AYS17.

The following is a simple assembly language program along with its assembly encoded string.

Example

Write a program to add up numbers 1 to 10.

```
    LD B, $10     ; Loads BR with positive integer 10
2   ADD B         ; Adds BR with the content of AR and puts the result in AR
    DEC B         ; Decrements the content of BR
    JNZ B, 2      ; Checks the content of BR and jumps to label 2 (ADD B) if BR is not zero
    HLT           : Halts and prepares the content of AR to show it correctly in JFLAP
```

Now **let's assemble** this program.

Encoding Assembly Programs

The object codes of all instructions are **prefixed with '#'** as the instructions separator.

Example (cont'd)

Instructions	**Assembly Codes**
LD B, $10	#LBV$1111111111
2 ADD B	#11AB
DEC B	#DB
JNZ B, 2	#ZB11
HLT	#H

All parts of the instructions codes are concatenated and there is **NO space** between them. Therefore, the object code of the above program that should be put on the tape is:

#LBV$1111111111#11AB#DB#ZB11#H

Technical Notes

1. The **output** of the program should go to AR and that's the only thing that matters in this project.
2. This version of AYS17 is for numerical operations and it cannot process strings.
3. You might use "S" (= stay option), block feature, variable assignments, and JFLAP's special characters '!' and '~'. These are great features that tremendously facilitate the design process.
   For more information, refer to the JFLAP's documentations and tutorials.
4. In JFLAP preferences → Turing Machine Preferences: uncheck "Accept by Halting" and check the rest before writing and testing your code.
5. Test your Turing machine as a transducer.
6. The look of the design is important for debugging purpose. Therefore, organize your design in such a way that it shows different modules clearly. Also, document very briefly your design by using JFLAP notes on the TM design page.

Rubrics

- I'll test your design with 10 programs and you'll get +20 for every success pass (**200 points**).
- If your code is not valid (e.g. there is no initial state, it is implemented by JFLAP 8, or so forth) you'll get 0 but you'd have chance to resubmit it with -20% penalty.
- You'll get -10 for wrong filename!
- Note that if you resubmit your assignment several times, Canvas adds a number at the end of your file name. I won't consider that number as the file name.

What you submit?

1. Design and test your program by the provided JFLAP
2. Save it as: Team_Name.jff
   (e.g.: SJSUAwesome.jff)
3. Upload it in the Canvas before the due date.

General Notes

- This is a team-based project.
- Teams can share "**test programs**" via Canvas discussion.
- For late submission policy, please read the greensheet.
- Always read the requirements at least 10 times! It is unacceptable if an engineer is not accurate enough.
- If there is any question or concern, please open a discussion in Canvas.