CS220 Computer Organization

Spring 2012

Machine Problem 2

Due February 8, 2012

Objectives

With machine problem 2 you will build upon the bouncing light program from MP1 to create the video game Pong1D, a one-dimensional version of the classic Pong. You will learn basic input/output operations using a programming style called *polling*, the simplest style of I/O. You will learn to use the PAL8 assembler—no more translating programs into machine language. You will write the program yourself, rather than simply translating a program that I gave you.

The basic requirement for MP2 is to create Pong1D so that a “ball” (AC light) can be served, and then players can hit it back and forth until the ball is missed, at which point the ball can be served again. For those who want a greater challenge, keep score and show the score using lights in MQ. For those who want an extreme challenge, keep score and print the score in decimal on the Teletype (TTY) after each point.

Pong1D

In Pong1D the “ball” is a light in AC that moves left and right at a speed determined by the low 6 bits of the switch register, as in MP1. A player tries to hits the ball by pressing any key on the TTY. If the key is pressed when the ball is at the left or right end, it’s a hit and the volley continues with the ball reversing direction. If no key is pressed when the ball is at the left or right end, it’s a miss. Note that a miss can result from a key being pressed too soon, too late, or not at all. When the ball is missed the TTY bell is rung, the other player scores a point, and the game waits for the next serve. A serve occurs when the “S” key is pressed on the TTY, with the ball starting in the opposite direction from what it was when the ball was missed.

For the advanced versions of Pong1D that keep score, when the game is waiting for a serve the “C” key can be pressed to clear the score to 0.

Readings

Important: The following are reference documents, not tutorials. Read them once through for familiarity, and then refer back as needed while working on MP2, but don’t expect everything to make perfect sense on a first reading. You will learn what you need through a combination of class meetings, readings, and the practice that will come with working the machine problem.

- *Small Computer Handbook* (1970): Pages 71 – 74 on the IOT instructions for reading characters from the Teletype (TTY) keyboard and printing characters on the TTY printer. Page 365 is a good reference summary of the TTY IOT instructions.

- *OS/8 Handbook* (1974) Chapter 3: Pages 3-5, starting at “Character Set”, to 3-30, ending at “Conditional Assembly Pseudo-Operators”, but skipping the sections “Symbol Table”, “Internal Symbol Representation for PAL8”, “Autoindexing”, “Extended Memory”, and “End-Of-File”. This is DEC’s documentation of the PAL8 assembler.

- *PDP-8 Sim*: This explains operation of the simulator. Pay particular attention to “PDP-8 Documents”, “ASR33 Teletype”, and “Hardware Breakpoint”.

- *PAL8S*: This explains how to use the PAL8 assembler that comes with the PDP-8/I simulator, and the differences between PAL8 as implemented and DEC’s original.

<u>Source Code</u>

For MP2 I will post two PAL8 source files:

- `bounce.pal8` is the source code for MP1, so you don’t have to type it in. Parts of it will be useful for MP2. Note that I have made some slight simplifications using features of the assembler: `mask` is replaced with a literal, and `pwait` is removed because the assembler will automatically make an indirect address.

- `echo.pal8` is a simple program that reads characters from the TTY keyboard and echoes then to the TTY printer. It illustrates the basic operation of TTY IOT instructions.

<u>Procedure</u>

If you have installed the simulator on your own computer, get the latest version as instructed in the *PDP-8/I Simulator* section on Blackboard. There are some enhancements and bug fixes that you might need. If you are using lab (224) computers, I will install the latest version by 5:00 Wednesday Feb 1.

Load and run the two PAL8 source files (one at a time) to confirm that you know how to use the assembler, and how to operate the simulated ASR33 Teletype.

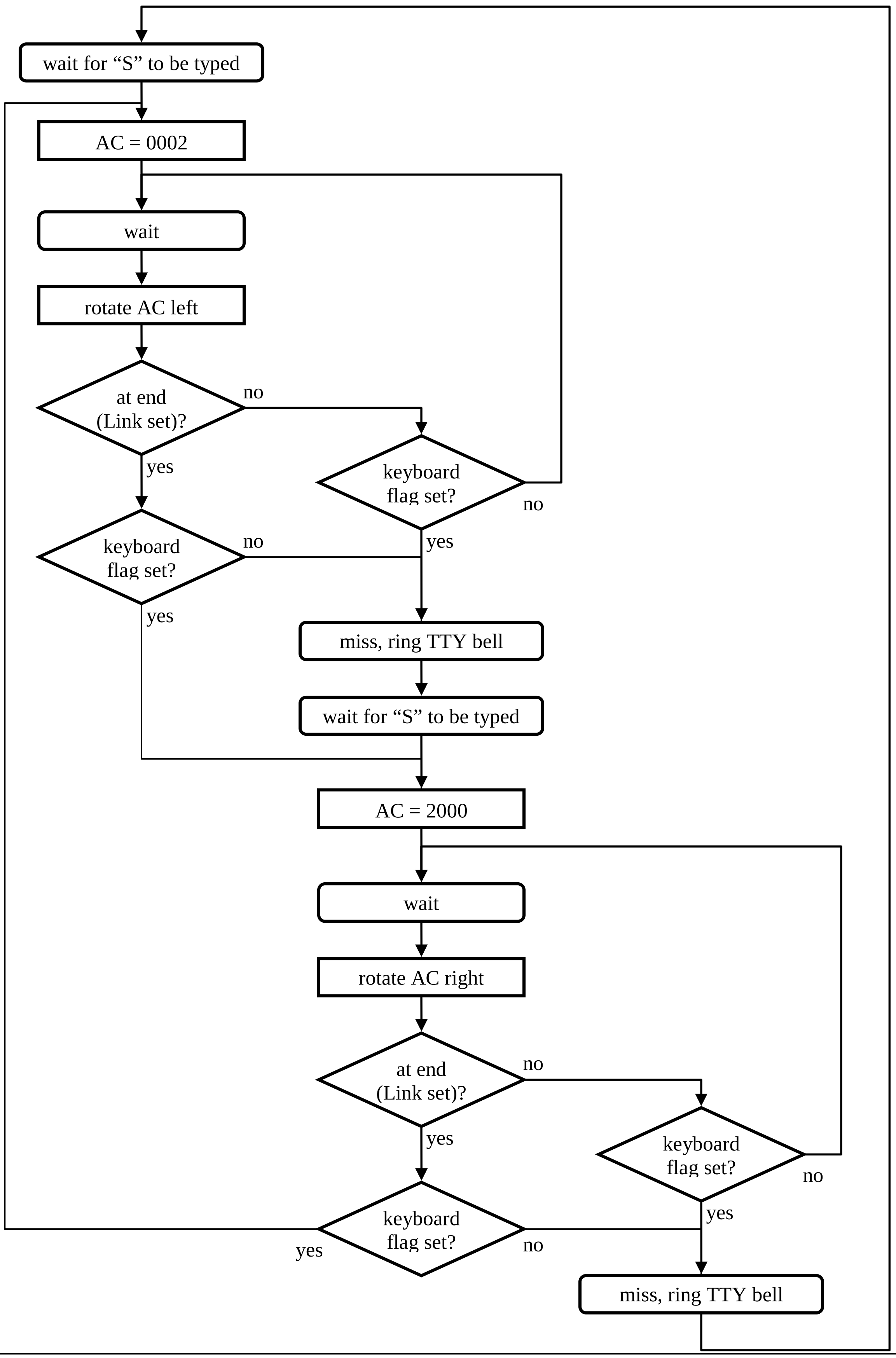
Write and debug your program. You can use the suggested flowchart below, or design it however you like. Note that in the flowchart, the rounded rectangles should be subroutine calls. The flowchart duplicates lots of logic for the left-moving and right-moving loops, which is easy to code but not the best programming style. My version has a single loop that can move either way, but figuring out how to do that cleanly is pretty advanced.

Note that you can ring the Teletype bell by printing the character ^G (control-G), ASCII code 207 octal.

<u>Deliverables (Blackboard)</u>

- Your PAL8 source files.

- A PDP8 document file saved with your program running.


wait for “S” to be typed
AC = 0002
wait
rotate AC left
at end (Link set)?
no
yes
keyboard flag set?
no
yes
keyboard flag set?
no
yes
miss, ring TTY bell
wait for “S” to be typed
AC = 2000
wait
rotate AC right
at end (Link set)?
no
yes
keyboard flag set?
no
yes
keyboard flag set?
yes
no
miss, ring TTY bell