**Wagner Baseball using 8051 Microcontroller**

2018 Microprocessor Team Project Report

Group 19

CHO YUN SANG

CHOI YUNG MIN

HONG SUK WOO

Dept. of Electronic Engineering

Hanyang University

**I. Objective**

The objectives of this team project are applying our knowledge learnt in class and develop our first assembly language program.

This program will be a baseball game using LCD and LED display, Dot matrix, key pad, Segments array, Buzzer and Motor.

**II. Individual Roles**

JO YUN SANG:

1. Make several button and its function(Single, Double, Triple, HomeRun)
2. Make several features ( Blink, Buzzer)

CHOI YUNG MIN:

1. Make algorithm to make decimal numbers from hexadecimal numbers
2. Expand the number of base from 1 to 3
3. Modify the ‘WALK’ function

HONG SUK WOO

1. Make several features (LED, MOTOR)
2. Make Hangul Database

**III. Implementation**

**Problem 1. Fix the score above 10**

In program, we use variables named ‘Score-Player1’, ‘Score-Player2’ for each player’s score. These variables are initially hexadecimal number. So, we have to convert these variables to decimal numbers. The Following is the code we used.

|  |
| --- |
| Score-Player1 EQU 35H ; The score of Player 1  Score-Player2 EQU 36H ; The score of Player 2  IncreaseScore:  MOV A, Player-At-Bat  CJNE A, #1, IncPlayer2Score ;check which player is playing  IncPlayer1Score:  MOV A, Score-Player1  INC A ;increase player1’s score by one  CALL MAKEDECIMAL ;Call function which makes decimal number  MOV Score-Player1, A  JMP FinishScoreInc  IncPlayer2Score:  MOV A, Score-Player2  INC A  CALL MAKEDECIMAL  MOV Score-Player2, A  FinishScoreInc:  CALL IncScoreEvent  MOV A, #0  MOV Ball-Count, #0  MOV Strike-Count, #0  CALL DisplayScores  RET  MAKEDECIMAL:  MOV R1, A ;move player`s score to R1  ANL A, #00001111B ;clear the upper 4 bits  CJNE A, #00001010B, MAKERET ;if lower 4bits aren`t equal to 10, jump to MAKERET  MOV A,R1 ;move R1 to A  ANL A, #11110000B ;clear lower 4bits  ADD A, #00010000B ;add 1 in second digit for LCD  RET  MAKERET:  MOV A,R1  RET |

In this code, we make MAKEDECIMAL function to make decimal number. In function, upper 4bits are equal to second digit number in decimal and lower 4bits are equal to first digit number in decimal.

So, we check score variable and if it is same as xxxx1010B, we clear lower 4bits and add one in upper 4bits. This process can make hexadecimal to decimal.

**Problem 2. Add second and third bases**

In the early program, runner can only go to first base. So, we fixed this and make runner can go to second and third base. The following code is code for Walk case with second, third base.

|  |
| --- |
| Base1-Occupied EQU 3CH ; 1 if First Base is Occupied, O if not  Base2-Occupied EQU 3DH ; 1 if Second Base is Occupied, O if not  Base3-Occupied EQU 3EH ; 1 if Third Base is Occupied, O if not  WalkBatter:  MOV A, Base1-Occupied  CJNE A, #1, OccupyBase1 ;check if base1 is occupied  JMP OccupyBase2  OccupyBase1: ;if base1 is not occupied,  MOV Base1-Occupied, #1 ;make base1 occupied  JMP COPYBASE  OccupyBase2: ;if base1 is occupied  MOV A,Base2-Occupied  CJNE A, #0, OccupyBase3 ;check if base2 is occupied  MOV Base2-Occupied, #1 ;if not, make base2 occupied  JMP COPYBASE  OccupyBase3: ;if base2 is occupied  MOV A,Base3-Occupied  CJNE A, #0, INCSCORE ;if base3 is occupied, increase score  MOV Base3-Occupied, #1 ;if not, make base3 occupied  JMP COPYBASE  INCSCORE:  CALL IncreaseScore  COPYBASE:  CALL CopyBasesToDotM ;call function to display runner on Dot-Matrix  RET |

we used three variables for each base occupation. for each base, we checked each base is occupied already, if so we move to next base and do the same thing. If not, we just make that empty base occupied. doing so, we can display the decimal score on LCD.

In addition, we tried to display runners on Dot-Matrix using following code. We just paste codes for base 3 for simplicity.

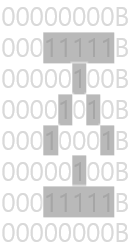
|  |
| --- |
| CopyBasesToDotM:  CALL CopyBase1ToDotM  CALL CopyBase2ToDotM  CALL CopyBase3ToDotM  RET  CopyBase3ToDotM: ; Check if base 3 is occupied  MOV A, Base3-Occupied  CJNE A, #0, AddBase3 ; If so, jump to AddBase3  JMP RemoveBase3 ; If not,jump to RemoveBase3  AddBase3:  CALL WriteBase3Red  RET  RemoveBase3:  CALL EraseBase3Red  RET  WriteBase3Red: ; Write Base3 Red-Dot-Matrix  MOV A, #Dot-Red-Matrix ; Move Dot-Red-Matrix to A  ADD A, #Base2Row ; Add Base2Row to A (Base2Row = Base3Row)  MOV R1, A ; A is 52H now  MOV A, @R1  ORL A, #Base3Bits ; Set base3 column bit  MOV @R1, A  RET  EraseBase3Red: ; Erase Base3 Red-Dot-Matrix  MOV A, #Dot-Red-Matrix  ADD A, #Base3Row ; Move to 52H which is Base3 Row address  MOV R1, A  MOV A, @R1  CPL A  ORL A, #Base3Bits  CPL A ; Clear base3 column bit  MOV @R1, A  RET |

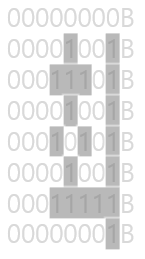
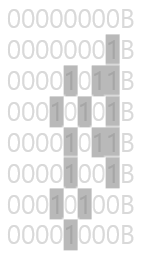
To display runner on Dot-Matrix, We check base-Occupied variables and Write or Erase base Red-Dot-Matrix. To do that, we go to Base Row Address and Write or Erase base column bit.

**Add group members names in Hangul for Players 1 and 2.**

First we make database of our name in Hangul.

As Following picture, we make ‘조’,’윤’,’상’,’ ‘최’,’영’,’민’ by 0 and 1

After making these database, We linked these to functions like ShowMsgPlayer. Here is following code

|  |
| --- |
| DisplayAtBat:  MOV A, Player-At-Bat  CJNE A, #1, Player2AtBat  CALL ShowMsgPlayer  CALL ShowMsgYunSang  RET  Player2AtBat:  CALL ShowMsgPlayer  CALL ShowMsgYungMin  RET  ShowMsgPlayer:  CALL ClearTopMsg  CALL MoveCurTopLeft  MOV DPTR, #MessagePlayer  MOV Message-Lo-Byte, DPL  MOV Message-Hi-Byte, DPH  MOV Message-Length, #5  CALL WriteMsgToLCD  RET  ShowMsgYunSang:  CALL StartHangul  CALL InstallYunSang  CALL MoveCurTopRight  CALL PrintYunSang  ShowMsgYungMin:  CALL StartHangul  CALL InstallYungMin  CALL MoveCurTopRight  CALL PrintYungMin  RET  InstallYunSang:  CALL Install-JO  CALL Install-YUN  CALL Install-SANG  RET  InstallYungMin:  CALL Install-CHOI  CALL Install-YUNG  CALL Install-MIN  RET  PrintYunSang:  MOV R1,#Char-Code-JO  CALL WriteR1CharToLCD  MOV R1,#Char-Code-YUN  CALL WriteR1CharToLCD  MOV R1,#Char-Code-SANG  CALL WriteR1CharToLCD  RET  PrintYungMin:  MOV R1,#Char-Code-CHOI  CALL WriteR1CharToLCD  MOV R1,#Char-Code-YUNG  CALL WriteR1CharToLCD  MOV R1,#Char-Code-MIN  CALL WriteR1CharToLCD  RET |

**Add some additional buttons**

First, we add several keypad constants like following code.

|  |
| --- |
| SingleKeyPress EQU 04H ; The Key-Number if Single is pressed  DoubleKeyPress EQU 05H ; The Key-Number if Double is pressed  TripleKeyPress EQU 06H ; The Key-Number if Triple is pressed  HomeRunKeyPress EQU 07H ; The Key-Number if HomeRun is pressed  FoulKeyPress EQU 08H ; The Key-Number if Foul is pressed |

Second, we expand the ‘Check Key-press Function’ to check these keys.

CheckWalk: CJNE A, #WalkKeyPress, CheckSingle

CALL WalkAction

JMP FinishKeyCheck

CheckSingle: CJNE A, #SingleKeyPress, CheckDouble

CALL SingleAction

JMP FinishKeyCheck

CheckDouble: CJNE A, #DoubleKeyPress, CheckTriple

CALL DoubleAction

JMP FinishKeyCheck

CheckTriple: CJNE A, #TripleKeyPress, CheckHomeRun

CALL TripleAction

JMP FinishKeyCheck

CheckHomeRun: CJNE A, #HomeRunKeyPress, CheckFoul

CALL HomeRunAction

JMP FinishKeyCheck

CheckFoul: CJNE A, #FoulKeyPress, FinishKeyCheck

CALL FoulAction

JMP FinishKeyCheck

FinishKeyCheck:

CALL DisplayAll

RET

After adding a code to activate the keypad, we created an algorithm for each function.

The ‘Walk’ function is already done of **Problem 2** as named ‘Walkbatter’, So we have to make Single, Double, Triple, Homerun, and Foul action.

**1) Algorithm for Single**

We considered the following points.

i) Single is like a Walk

ii) However, the base 2 and base 3 would be shifted even when base1 is empty

So we constructed algorithms by dividing the number of cases whether there were no runners on second and third base

Here is the code.

SingleAction:

CALL ShowMsgSingle

MOV A,Base3-Occupied

CJNE A,#1,SingleAction2

CALL IncreaseScore

MOV Base3-Occupied, #0

MOV A,Base2-Occupied

CJNE A,#1,SingleAction3

MOV Base2-Occupied, #0

MOV Base3-Occupied, #1

CALL MakeWalk

CALL CopyBasesToDotM

RET

SingleAction2:

MOV A,Base2-Occupied

CJNE A,#1,SingleAction3

MOV Base2-Occupied, #0

MOV Base3-Occupied, #1

CALL MakeWalk

CALL CopyBasesToDotM

RET

SingleAction3:

CALL MakeWalk

RET

**2) Algorithm for Double**

We considered the following points.

i) Double is like to perform Walk twice

ii) However, the base 1 must be erased after the twice Walk

iii) Also, a runner in base 2 and 3 should be moved (A runner in base 1 would be moved naturally by ‘Walk’ function

So, Like Single action, we constructed algorithms by dividing the number of cases

Here is the code.

DoubleAction:

CALL ShowMsgDouble

MOV A,Base1-Occupied

CJNE A, #1, DoubleAction2

CALL MakeWalk

CALL MakeWalk

MOV Base1-Occupied, #0

CALL CopyBasesToDotM

RET

DoubleAction2:

CALL MakeWalk

CALL MakeWalk

MOV A,Base3-Occupied

CJNE A, #0, DoubleAction3

DoubleActionback:

MOV Base1-Occupied, #0

CALL CopyBasesToDotM

RET

DoubleAction3:

CALL IncreaseScore

MOV Base3-Occupied, #0

JMP DoubleActionback

**3) Algorithm for Triple**

We considered the following points.

i) Single is similar to perform Walk 3 times

ii) However, the base 1 and 2 must be erased after the three-times Walk

Unlike Single and Double action, since the all previous runners would be get Homeplate, we just have to perform Walk 3 times and erase base 1,2

Here is the code

TripleAction:

CALL ShowMsgTriple

CALL MakeWalk

CALL MakeWalk

CALL MakeWalk

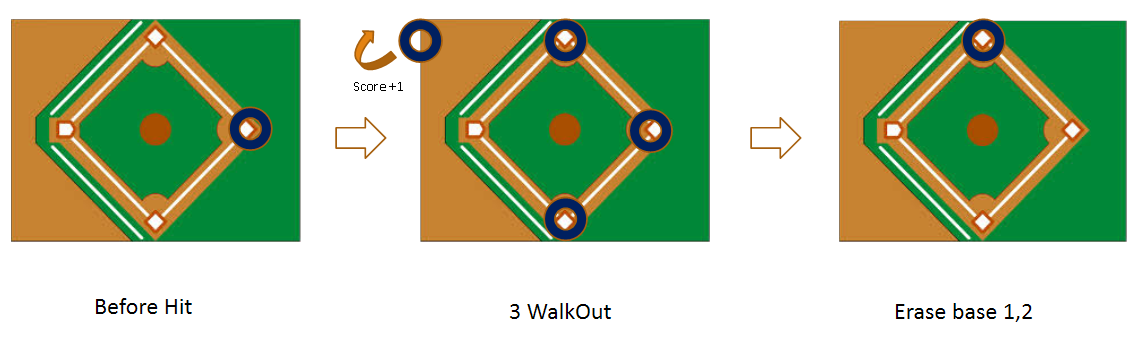
MOV Base1-Occupied, #0

MOV Base2-Occupied, #0

CALL CopyBasesToDotM

RET

Here is a picture showing the concept of a Triple



**4) Algorithm for Homerun**

We considered the following points.

i) Single is similar to perform Walk 4 times

ii) However, the base 1,2, and 3 must be erased after the three-times Walk

Like Triple action, we just have to use Walk and Erase

Here is the code

HomeRunAction:

CALL ShowMsgHomeRun

CALL MOTOR

CALL MakeWalk

CALL MakeWalk

CALL MakeWalk

CALL MakeWalk

MOV Base1-Occupied, #0

MOV Base2-Occupied, #0

MOV Base3-Occupied, #0

CALL CopyBasesToDotM

RET

**5) Algorithm for Foul**

We considered the following points.

i) Foul is the increasing-strike count function

ii) However, the strike count does not increased by foul if it is already 2 (this could be performed by using ‘CJNE’ instruction.)

Considering these point, we made following code.

FoulAction:

CALL ThrowStrike

CALL ShowMsgFoul

CALL FoulEvent

MOV A, Strike-Count

CJNE A,#2, IncStrikeAtFoul

RET

IncStrikeAtFoul:

INC A

MOV Strike-Count, A

RET

**Add 3 additional new features**

**1. LED Blinking and Buzzing Function**

In this baseball rules, there are several functions for LED

1. LED Blinking
   1. Upside blinking LED and downside blinking LED
   2. Odd LED blinking and even LED blinking
   3. Entire blinking
2. Buzzing Function

There is also Buzzing Function.

In FoulAction, there are ThrowStrike, ShowMsgFoul, FoulEvent components are called and we employ register A for strike-counting and do conditional jumping for IncStrikeAtFoul function, and then, we retire. After then, we define IncStrikeAtFoul function which could be elaborated by increasing A, moving striking-count function to A, and retiring. In FoulEvent, we move #05H memory address allocating to R3 register. In case of DisplayFoulEvent part, we are moving #10101010B LED command to register A, moving A’s logic to p1, calling EVENTDELAY function to visualize the LED function. Moving #01010101B LED command and then, we also employ EVENTDELAY function for altenative LED blinking. After then, we terminate by RET command.

|  |
| --- |
| FoulAction:  CALL ThrowStrike  CALL ShowMsgFoul  CALL FoulEvent  MOV A, Strike-Count  CJNE A,#2, IncStrikeAtFoul  RET  IncStrikeAtFoul:  INC A  MOV Strike-Count, A  RET  FoulEvent:  MOV R3,#05H  DisplayFoulEvent:  MOV A,#10101010B  MOV P1,A  CALL EVENTDELAY  MOV A,#01010101B  MOV P1,A  CALL EVENTDELAY  DJNZ R3,DisplayFoulEvent  RET |

There are WalkAction, WalkEvent, DiaplayWalkEvent, OutAction, OUTEVENT, OUTEVENTLED. In WalkAction, we call IncBall Count4, and WalkEvent components, and then, we retire. In WalkEvent, we allocate $05H to R3 register, and #11110000B and #00001111B LED commands to A register, and we each of them allocate EVENTDELAY function to visualize the LED blinking because without the function the showing of LED would be flipped out crossing our eyes. On OutAction, there are ThrowStriker, ShowMsgOut, IncOutCount, and OUTEVENT components, and retire. Moving #04H to R3 register, we are employing command to #11111111B and #00000000B commands to alternatively entirely blinking LED or none-blinking, so-called all-or-none blinking LED logic to the LED, and we each of them allocate EVENTDELAY function to visualize the LED blinking because without the function the showing of LED would be flipped out crossing our eyes.

|  |
| --- |
| WalkAction:  CALL IncBallCount4  CALL WalkEvent  RET  WalkEvent:  MOV R3,#05H  DisplayWalkEvent:  MOV A,#11110000B  MOV P1,A  CALL EVENTDELAY  MOV A,#00001111B  MOV P1,A  CALL EVENTDELAY  DJNZ R3,DisplayWalkEvent  RET  OutAction:  CALL ThrowStrike  CALL ShowMsgOut  CALL IncOutCount  CALL OUTEVENT  RET  OUTEVENT:  MOV R3,#04H  OUTEVENTLED:  MOV A,#11111111B  MOV P1,A  CALL EVENTDELAY  MOV A,#00000000B  MOV P1,A  CALL EVENTDELAY  DJNZ R3,OUTEVENTLED  RET |

**2. Operate Buzzer when Inning is changed**

Because the address of motor in 8051 kit is 0FFEFH, we connect DPTR to this address, and operate buzzer to input #10000000B in this address during delay time, and we stopped it by inputting #00000000B.

Here is the code

EndInningEvent:

MOV DPTR, #BUZZER

MOV A,#10000000B

MOVX @DPTR,A

CALL EVENTDELAY

MOV DPTR, #BUZZER

MOV A,#00000000B

MOVX @DPTR,A

RET

And we call this function in FinishInning action like following code

FinishInningInc:

MOV Inning-Number, A

CALL DisplayInningNum

CALL EndInningEvent

RET

**3. Blink Segments array when score is increased**

By switching to display actual scores and empty scores, we could perform ‘Blink’ segments array.

Here is the code

IncScoreEvent:

MOV R3,#05H

IncScoreArray:

CALL DisplayScores

CALL EVENTDELAY

MOV A, #0FFh

MOV DPTR, #LED-7-Segment1

MOVX @DPTR, A

MOV DPTR, #LED-7-Segment2

MOVX @DPTR, A

MOV DPTR, #LED-7-Segment3

MOVX @DPTR, A

CALL EVENTDELAY

DJNZ R3,IncScoreArray

RET

And we call this function in Finnish Score action like following code

FinishScoreInc:

CALL IncScoreEvent

MOV A, #0

MOV Ball-Count, #0

MOV Strike-Count, #0

CALL DisplayScores

RET