|  |
| --- |
| Dalhousie University  Department of Electrical and Computer Engineering |
| ECED 3403 Computer Architecture |
| Assignment #4 – Conditional Execution |
| Prepared by  UGOCHUKWU CHUKWU, B00556842 |
|  |
|  |
| **31 July, 2014** |

DESIGN DESCRIPTION

**Introduction**  
The Z8 emulator with cache has been updated to incorporate a conditionally executed statement; i.e IF instruction to speed up the machine by reducing the number of predictions and jumps done by the system. This is especially ideal for the pipeline machine as it saves time, the machine would use to bubble up stages in the pipeline due to a wrong prediction/ branch taken.

When an IF instruction is encountered, the machine fetches the next byte that holds the condition code, true and false instruction count. Furthermore, the true instructions are executed, and the false instructions are skipped when the condition code is true; otherwise when the condition is false the execution of the machine is vice versa. The restrictions on the conditional statements is that they do not change the state of the system; i.e the FLAGS, however if a control of flow instruction is encountered and successfully takes place the conditional statements should cease to execute.

In our design, we do not account for how the emulator should act when an interrupt occurs during the execution of a conditional statement, the two possible solutions could be to cease from executing further and skip over all the conditional statements, or to push the pc, tcount and fcount unto the stack, and pull them back in an IRET for the computer to successfully return to its previous state of operation.

BEHAVIOURAL DESCRIPTION

IF IS DECODED

FETCH NEXT BYTE

EXTRACT HNIB AND LNIB

HNIB IS CONDITION CODE

EXTRACT MOST SIGNIFICANT BITS AND LEAST SIGNIFICANT BITS FROM LOWER NIBBLE

MOST SIGNIFICANT IS TCOUNT, LEAST SIGNIFICANT IS FCOUNT

SAVE STATE, STORE FLAGS IN TEMPFLAGS

CEXEC = CHECK CONDITION CODE

/\* IF TRUE CEXEC IS 1, ELSE CEXEC IS 0 \*/

/\* AT THE END OF EACH INSTRUCTION CYCLE \*/

IF CEXEC IS 1 /\* EXECUTE TRUE PART OF COND STATEMENT \*/

IF TCOUNT IS GREATER THAN 0

DECREMENT TCOUNT

RESTORE STATE, FLAGS <-- TEMPFLAGS

/\* ALLOW EXECUTION TO CONTINUE \*/

ENDIF

/\* TRUE PART IS DONE EXECUTING \*/

ELSE IF FCOUNT IS GREATER THAN 0

/\* SKIP FALSE PART \*/

WHILE FCOUNT IS GREATER THAN 0

FETCH NEXT INSTRUCTION FROM PROG\_MEM

INCREMENT PC BY OPCODE\_SIZE BYTES

DECREMENT FCOUNT BY 1

ENDWHILE

ENDELSE

RETORE STATE

ENDIF

ELSE /\* CEXEC IS 0

EXECUTE FALSE PART OF COND STATEMENT \*/

IF TCOUNT IS GREATER THAN 0

WHILE TCOUNT IS GREATER THAN 0

FETCH NEXT INSTRUCTION

INCREMENT PC BY OPCODE\_SIZE BYTES

DECREMENT TCOUNT BY 1

ENDWHILE

ENDIF

ELSE IF FCOUNT IS GREATER THAN 0

DECREMENT FCOUNT

RESTORE STATE, FLAGS<--TEMPFLAGS

ENDELSE

RESTORE STATE

ENDELSE

/\*\*\*\*\*\*\*\*\*\*\*\*\*CHANGES WERE MADE TO THE JP, CALL INSTRUCTIONS\*\*\*\*\*\*\*\*\*\*/

TO ALLOW THESE INSTRUCTIONS TO SUCCCESSFULLY CHANGE THE FLOW OF CONTROL WE ENABLED THESE INSTRUCTION STO FORCE THE TCOUNT AND FCOUNT TO ZERO IF THEY ARE SUCESSFULLY EXECUTED IN A CONDITIONAL STATEMENT.

DATA DICTIONARY

Tcount = 2{BITS}2 \*holds the number of true instructions to carry out\*

Fcount = 2{BITS}2 \*holds the number of false instructions to carry out\*

Opcode\_size = BYTE \*number of operands bytes for the opcode\*

Tempflags = BYTE \* maintains the original state of the system during the execution of a conditional statement\*

Cexec = [0|1] \* determinant if the condition code is true – 1/ false 0 \*