סימולטור למעבד Inorder Pipelined

מדריך למשתמש

טבלת שינויים

מס' שינוי	תאור השינוי	מותאם לגרסא	תאריך	אחראי
1	פתיחת המסמך, גרסא ראשונית	1.0	21.01.2017	גרי סלבין & מיכאל באגאטי
2				
3				

תוכן עניינים

1. הקדמה

סימלטור זה נבנה בתור פרוייקט ב' של סטודנטים מן הפקולטה.

מטרתו היא לאפשר **לכם**, הסטודנטים להנדסת חשמל להעמיק בנושא התכן הלוגי באמצעות הדגמה של פעולת מעבד MIPS מסוג Pipeline שעליו אתם לומדים בקורס הנ"ל.

כידוע לכם, כמו בכל פרוייקט או בכל תוכנה עלולים לצוץ בעיות שאנחנו, המפתחים, לא ציפינו להם. לכן, נשמח תוכלו להפנות את תשומת לבנו (או במילים אחרות, תשומת לב אחראי המעבדה לVLSI) בכל באג שתמצאו בגרסא הנוכחית כדי שנוכל לתקן אותו במהרה.

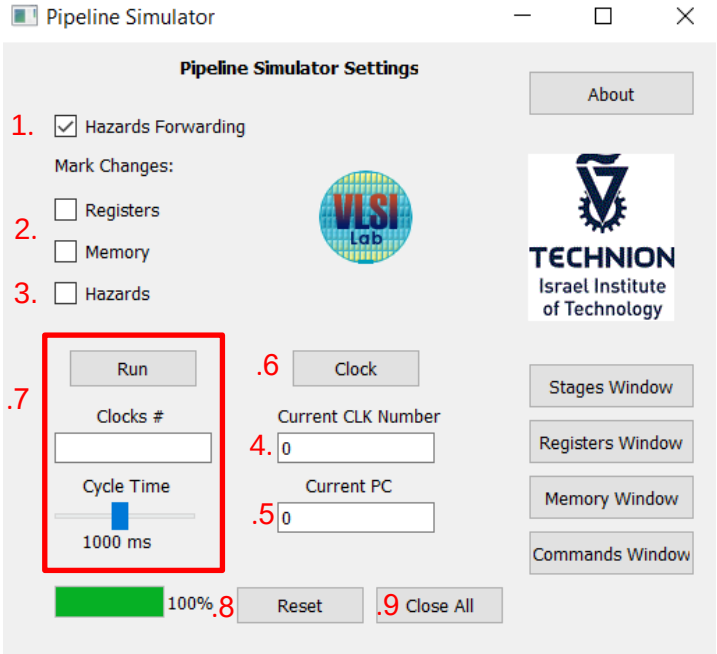
<u>למתעניינים:</u> הפרוייקט נבנה באמצעות 2 כלים עיקריים:

Visual Studio 2017 Community - שבה נכתב הקוד בשפת ++C.

QT 5.9.1 – הממשק הגרפי שאיתו עוצבה התוכנית.

2. אופן שימוש

- פתיחת הסימולטור: לתחילת הסימולטור יש לפתוח את קובץ ההרצה MIPS Simulator.exe. כעת יפתח החלון הראשי של הסימולטור.



החלון הראשי מציג את כל אפשרויות ההפעלה של הסימולטור וכן כפתורים לפתיחת חלונות נוספים.

<u>אפשרויות סימון ותצוגה:</u>

1. hazard forwarding – בעת סימון אפשרות זאת הסימולטור ישתמש במנגנון טיפול בHazards על ידי forwarding unit. במידה ולא יסומן, הטיפול בHazards יהיה על ידי דחיפת פקודות NOP בין הפקודות שביניהן יהיה Hazard.

2. Registers/Memory – כאשר תיבה זאת מסומנת, במהלך ריצת הסימולטור יסומנו שינויים ברגיסטרים/בזיכרון (בהתאם לסימון) בחלון המתאים. כאשר תיבה זאת לא מסומנת, השינויים לא יצבעו בעת ביצוע.

3. Hazards – בעת סימון תיבה זאת ייצבעו פקודות שביניהן קיים Hazard בחלון stages window.

4. Current CLK Number – מציג את מחזור השעון הנוכחי.

5. Current PC – מציג את ה Program Counter הנוכחי.

אפשרויות הרצה:

6. Clock – מריץ מחזור שעון אחד של התכנית.

7. Run & Clocks # & Cycle Time - במידה והמשתמש רוצה להריץ מספר מחזורי שעון ברצף, יכניס המשתמש מספר מחזורי שעון להרצה בחלונית # Clocks וכן יבחר את זמן הריצה של כל מחזור (אם רוצים לעקוב אחר הפקודות שמתבצעות והכתיבות לרגיסטרים וזיכרון יש לבחור זמן מספיק ארוך בכדי שיהיה ניתן לשים לב לשינויים). לאחר בחירת מספר המחזורים וזמן הריצה יש ללחוץ על כפתור Run.

8. Close All – סוגר את כל החלונות הפתוחים פרט לחלון הראשי.

9. Reset – מאפס את התוכנית (PC, רגיסטרים, זיכרון, מחזור שעון).

חלונות:

10. About – מכיל מידע על התוכנה.

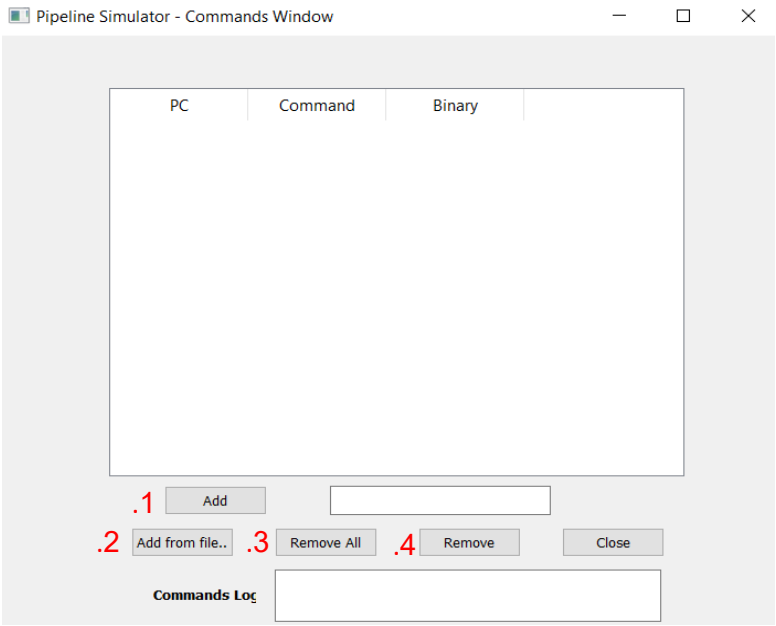
11. Stages Window – מכיל שרטוט סכמתי של התכן ואת השלבים של המעבד.

12. Registers Window – מכיל את זיכרון הרגיסטרים.

13. Memory Window – מכיל את הזיכרון הפיזי.

14. Commands Window – מכיל את זכרון הפקודות.

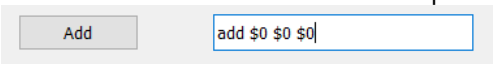
- הטענת זכרון הפקודות: כעת יש לטעון את הפקודות הרצויות לזכרון הפקודות באמצעות חלון זכרון הפקודות. יש ללחוץ על כפתור "Commands Window".



חלון זה שולט על הוספת והסרת פקודות מהסימולטור. פורמט הפקודות צריך להיות כפי שמופיע בנספח. שגיאות בהכנסת הפקודות יוצגו בחלונית ה Commands Log.

אפשרויות הוספת פקודות:

1. הוספת פקודות בודדות מחלון הממשק. לדוגמא:



2. הוספת פקודות מקובץ – קובץ לדוגמא:

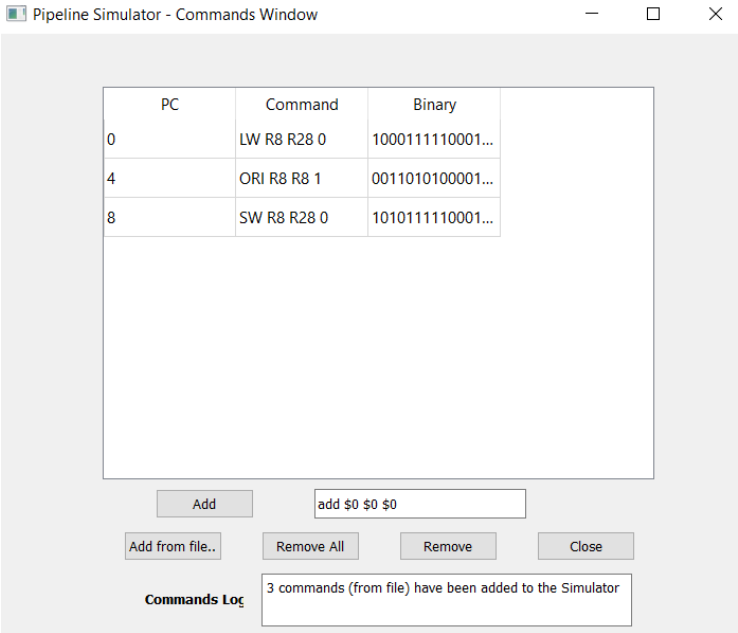
```
#C code:
 #if ( N%2 == 0 ) N++;

#assembly code:
  lw    $t0, 0($gp)        # fetch N
  ori   $t0, $t0, 1        # turn on low order bit
  sw    $t0, 0($gp)        # store result in N
```

יש לשמור על הכללים הבאים:

- שם הפקודה צריך להיות תקין ומהפקודות הנתמכות המופיעות בנספח.
- פרמטרי הפקודה מופרדים על ידי כל סוג של רווח או פסיק. כמובן שהפרמטרים צריכים להיות תקינים ולפי פורמט הפקודה כפי שמופיע בנספח.
- הערות יתחילו ב #. כל סימון אחר יגרור שגיאה.

לאחר הוספת קובץ הפקודות החלון ייראה כך:



ניתן לראות כעת את זיכרון הפקודות וכן בחלונית ה Commands Log ניתן לראות כי כל הפקודות נוספו בהצלחה.

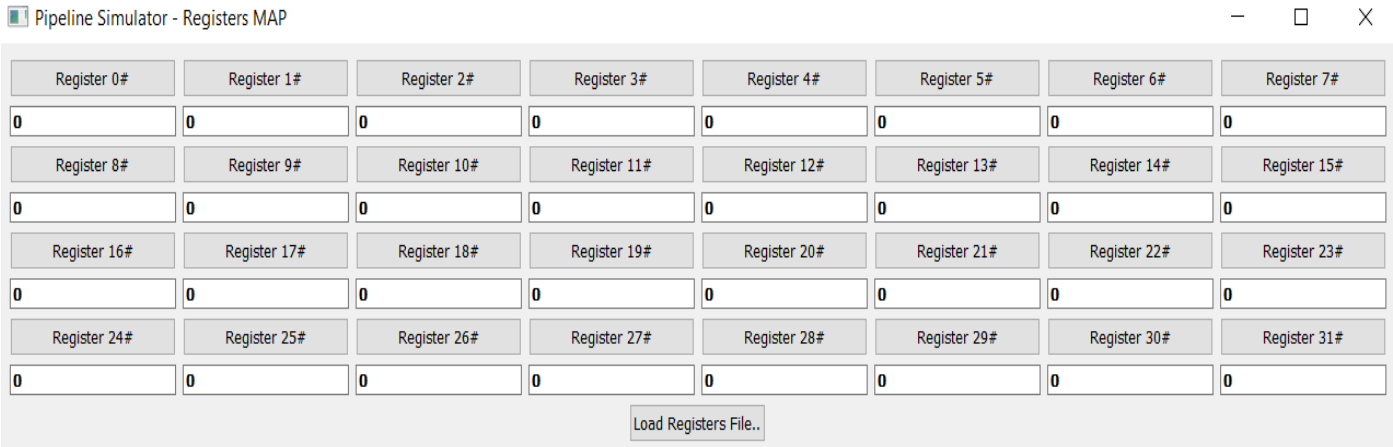
להסרת פקודות ישנן שתי אפשרויות:

3. Remove All – מסיר את כל הפקודות.

4. Remove – בעת סימון פקודה מסויימת בחלון, לחיצה על הפתור Remove יסיר את הפקודה המסומנת.

- הטענת שאר הזכרונות: כעת יש לטעון את מצב הרגיסטרים והזכרון ההתחלתיים (תוכלו לדלג על שלב זה אם תרצו להתחיל מרגיסטרים וזכרון מאופס). יש ללחוץ על "Registers Window" ו"Memory Window"

חלון Registers Window:

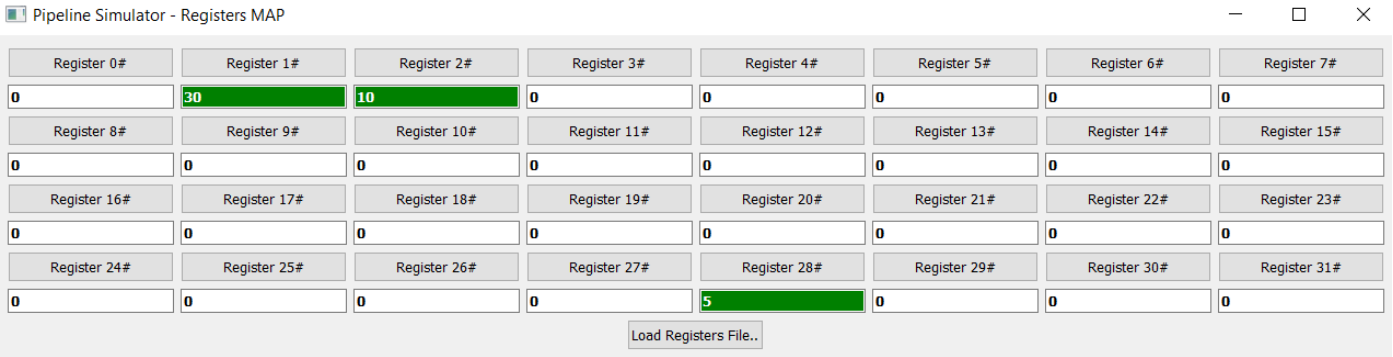


בחלון זה ניתן לראות את מצב הרגיסטרים בזמן ריצת הסימולטור. בתחילת התוכנית ניתן לטעון קובץ רגיסטרים על ידי לחיצה על הכפתור ...Load Registers File. על הקובץ להיות קובץ טקסט בפורמט CSV (הפרדה על ידי פסיק).

דוגמת קובץ רגיסטרים:

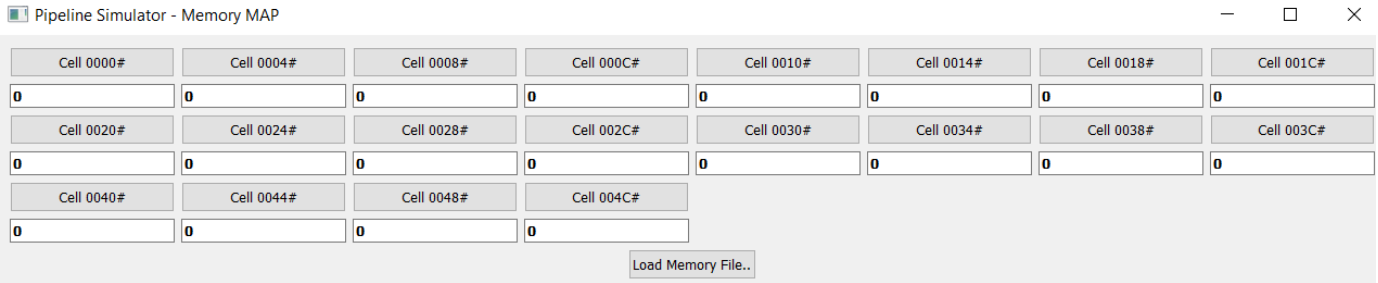
1,30
2,10
28,5

בטעינת קובץ טסט שיכיל את הדוגמא הנ"ל ייטען לרגיסטר 1 הערך 30, רגיסטר 2 ייטען לערך 10, ורגיסטר 28 לערך 5.
להלן חלון הרגיסטרים לאחר טעינת הקובץ לדוגמא:



רגיסטרים שייטענו לערך חדש בתחילת התוכנית ייצבעו בירוק. כמו כן במהלך ריצת הסימולטור, במידה חלונית הסימון Registers בחלון הראשי מסומנת, כל שינוי ברגיסטר ייצבע בירוק. בעת לחיצה על כפתור Reset בחלון הראשי כל הרגיסטרים יאופסו.

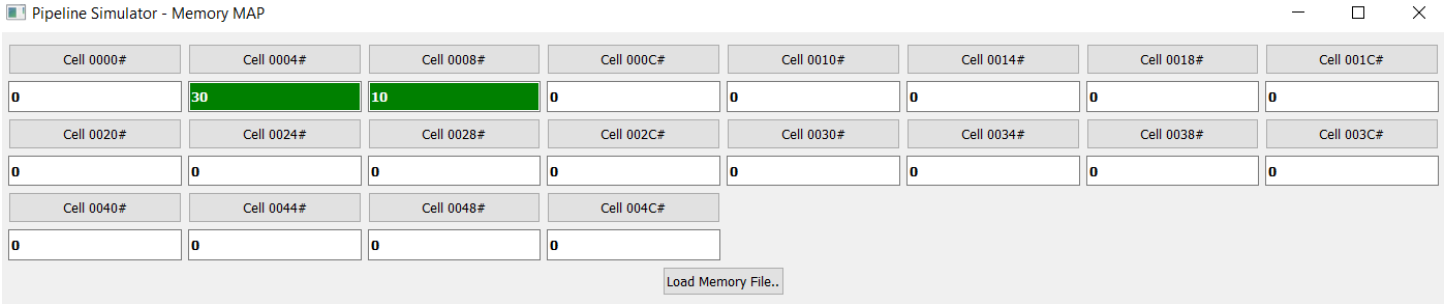
חלון Memory Window:



חלון זה דומה לחלון הרגיסטרים ומתנהג באותה צורה. לשם טעינת הזיכרון בתחילת התוכנית יש להזין גם כן קובץ בפורמט CSV כלהלן:

1,30
2,10

להלן תמונת הזיכרון לאחר טעינת הקובץ:



הספרה משמאל לפסיק מסמלת את תא הזיכרון, ומשמאל את הערך שייטען.
כמו כן הזיכרון כולו של הסימולטור הינו 4MB, אך לטובת התצוגה בסימולטור מוצג רק זיכרון חלקי ולכן שינויים שיבוצעו בתאי זיכרון שמחוץ לגודל הזיכרון שב GUI לא ייראו בממשק המשתמש.

שימו לב: לא ניתן לשנות את זכרון הפקודות/רגיסטרים/זכרון המרכזי במהלך ריצת התכנית אלא רק בפתיחת התכונה/לאחר אתחול התכנית (באמצעות כפתור Reset) !

- הרצת התכנית: כעת אפשר להתחיל להריץ את התכנית באמצעות אפשרויות ההרצה שפורטו בחלון המרכזי.

כעת ניתן לראות:

- כיצד סמן ה PC עובר בין הפקודות בחלון הפקודות (Commands Window)
- איך הפקודות עוברות במעבד משלב לשלב בחלון השלבים (Stages Window), והאם קיימים Hazards. (במידה והאפשרות סומנה)
- איך תאי זכרון/רגיסטרים משתנים בחלונות הזכרון (Memory Window) והרגיסטרים (Registers Window) בהתאם לריצת התכנית.

3. נספח רשימת פקודות חוקיות

Type :

Addition:

add rd rs rt

0	rs	rt	rd	0	32
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

example: add $1 $2 $3 -> R1 = R2 + R3

Subtract:

sub rd rs rt

0	rs	rt	rd	0	34
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

example: sub $1 $2 $3 -> R1 = R2 - R3

And:

and rd rs rt

0	rs	rt	rd	0	36
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

example: and $1 $2 $3 -> R1 = R2 & R3

Or:

or rd rs rt

0	rs	rt	rd	0	37
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

example: or $1 $2 $3 -> R1 = R2 || R3

Xor:

xor rd rs rt

0	rs	rt	rd	0	38
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

example: xor $1 $2 $3 -> R1 = R2 $\oplus$ R3

Nor:

nor rd rs rt

0	rs	rt	rd	0	39
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

example: add $1 $2 $3 -> R1 = R2 nor R3

Set less than:

slt rd rs rt

0	rs	rt	rd	0	42
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

example: slt $1 $2 $3 -> R1 = ((R2 < R3) ? 1 : 0)

Shift left logical:

sll rd rt shamt

0	rs	rt	rd	shamt	0
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

example: sll $1 $2 2 -> R1 = R2 << 2

Shift right logical:

srl rd rs shamt

0	rs	rt	rd	shamt	2
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

example: srl $1 $2 2 -> R1 = R2 >> 2

Shift right arithmetic:

sra rd rs shamt

0	rs	rt	rd	shamt	3
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

example: sra $1 $2 2 -> R1 = R2 >> 2

Shift left logical variable:

sllv rd rt rs

0	rs	rt	rd	0	4
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

example: sllv $1 $2 $3 -> R1 = R2 << R3

Shift right logical variable:

srlv rd rt rs

0	rs	rt	rd	0	6
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

example: srlv $1 $2 $3 -> R1 = R2 >> R3

Shift right arithmetic variable:

srav rd rt rs

0	rs	rt	rd	0	7
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

example: srlv $1 $2 $3 -> R1 = R2 >> R3

Jump and link register:

jalr rs rd

0	rs	0	rd	0	9
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

example: jalr $1 $2 ->

R2 = PC + 8 ; PC = (PC & 0xf0000000) | (R1 << 2)

Jump register:

jr rs

0	rs	0	0	0	8
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

example: jr $1 -> PC = (PC & 0xf0000000) | (R1 << 2)

: I-Type

And immediate:

andi rt rs imm

12	rs	rt	immidiate
6 bit	5 bit	5 bit	16 bit

example: andi $1 $2 2 -> R1 = R2 & 2

Or immediate:

ori rt rs imm

13	rs	rt	immidiate
6 bit	5 bit	5 bit	16 bit

example: ori $1 $2 2 -> R1 = R2 || 2

Xor immediate:

xori rt rs imm

14	rs	rt	immidiate
6 bit	5 bit	5 bit	16 bit

example: xori $1 $2 2 -> R1 = R2 $\oplus$ 2

Load upper immediate:

lui rt imm

15	0	rt	immidiate
6 bit	5 bit	5 bit	16 bit

example: lui $1 2 -> R1 = 2 << 16

addition immediate:

addi rt rs imm

8	rs	rt	immidiate
6 bit	5 bit	5 bit	16 bit

example: lui $1 $2 2 -> R1 = R2 + 2

Load word:

lw rt address

35	rs	rt	offset
6 bit	5 bit	5 bit	16 bit

example: lw $1 offset($2) 2 -> R1 = Memory[R2 + offset]

Store word:

sw rt address

43	rs	rt	offset
6 bit	5 bit	5 bit	16 bit

example: sw $1 offset($2) 2 -> Memory[R2 + offset] = R1

Branch on equal:

beq rs rt offset

4	rs	rt	offset
6 bit	5 bit	5 bit	16 bit

example: beq $1 $2 offset ->

advance_pc_to = ((R1 == R2) ? (offset) : (pc + 4))

Branch on not equal:

bne rs rt offset

5	rs	rt	offset
6 bit	5 bit	5 bit	16 bit

example: bne $1 $2 offset ->

advance_pc_to = ((R1 != R2) ? (offset) : (pc + 4))

Branch on less than equal zero:

blez rs offset

6	rs	0	offset
6 bit	5 bit	5 bit	16 bit

example: blez $1 offset ->

advance_pc_to = ((R1 <= 0) ? (offset) : (pc + 4))

Branch on greater than equal zero:

bgtz rs offset

7	rs	0	offset
6 bit	5 bit	5 bit	16 bit

example: bgtz $1 offset ->

advance_pc_to = ((R1 >= 0) ? (offset << 2) : (pc + 4))

: J-Type

Jump:

j target

2	Jump target
6 bit	26 bit

example: j target -> PC = (PC & 0xf0000000) | (target << 2)

Jump and link:

jal target

3	Jump target
6 bit	26 bit

example: jal target ->

R31 = PC + 8 ; PC = (PC & 0xf0000000) | (target << 2)

:NOP

Nop:

nop

0
32 bit

example: nop -> R0 = R0 << 0