

ORGANIZACIÓN DE COMPUTADORES: LABORATORIO 2

NICOLÁS OLIVARES

Profesores:

Felipe Garay

Erika Rosas

Nicolás Hidalgo

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS.....	v
ÍNDICE DE CUADROS	vi
CAPÍTULO 1. INTRODUCCIÓN.....	7
1.1 MOTIVACIÓN	7
1.2 OBJETIVOS	7
1.2.1 Objetivo general	7
1.2.2 Objetivos específicos	7
1.3 PROBLEMA	7
1.4 ORGANIZACIÓN DEL DOCUMENTO	8
1.5 HERRAMIENTAS	8
CAPÍTULO 2. MARCO TEÓRICO	9
CAPÍTULO 3. DESARROLLO.....	11
3.1 CONSTRUCCIÓN DE SIMULADOR	11
3.1.1 Instruction Fetch IF	12
3.1.2 Instruction Decode ID	13
3.1.3 Execution EX	13
3.1.4 Memory MEM	13
3.1.5 Write Back WB	13
3.2 RESULTADOS	13
3.2.1 Expr_aprox	14
Mejora 1	16
Mejora 2	19
3.2.2 Factorial	21
3.2.3 Suma_arreglo	22
3.2.4 Fizz_buzz	22
3.2.5 Fizz_buzz_div_2	23

CAPÍTULO 4. ANÁLISIS.....	25
4.0.1 Expr_aprox	25
Mejora 1	25
Mejora 2	25
4.0.2 Suma_arreglo	25
4.0.3 Factorial	25
4.0.4 Fizz_buzz	25
4.0.5 Fizz_buzz.div_2	25
CAPÍTULO 5. CONCLUSIÓN.....	27
CAPÍTULO 6. BIBLIOGRAFÍA.....	29

ÍNDICE DE FIGURAS

Figura 3-1: Pipeline de cinco etapas, base de la implementación[4]	12
Figura 3-2: Archivo de prueba expr_aprox	14
Figura 3-3: END.STATE de expr_aprox	15
Figura 3-4: HAZARDS de expr_aprox	16
Figura 3-5: expr_aprox con mejora 1	16
Figura 3-6: END.STATE de expr_aprox mejora 1	17
Figura 3-7: HAZARDS de expr_aprox mejora 1	18
Figura 3-8: expr_aprox con mejora 2	19
Figura 3-9: END.STATE de expr_aprox mejora 2	20
Figura 3-10: HAZARDS de expr_aprox mejora 2	21
Figura 3-11: HAZARDS de factorial sin mejora	22
Figura 3-12: END.STATE de fizz_buzz sin mejora	22
Figura 3-13: HAZARDS de fizz_buzz sin mejora	23
Figura 3-14: END.STATE de fizz_buzz_div_2 sin mejora	23
Figura 3-15: HAZARDS de fizz_buzz_div_2 sin mejora	24

ÍNDICE DE CUADROS

CAPÍTULO 1. INTRODUCCIÓN

1.1 MOTIVACIÓN

El uso de las herramientas computacionales actuales dentro de la vida cotidiana de cada persona, establece una necesidad de entender cómo funcionan los procesos que se ejecutan ajenos al ojo humano promedio dentro de una máquina de computo basada en el modelo de Von Neumann.

El lector de este informe de análisis podrá acceder a un conocimiento superior al que posee previo a la lectura, estableciendo en su mente el entendimiento de cómo las miles y millones de instrucciones que se ejecutan en un procesador, utilizando conceptos arquitecturales vistos en cursos de informática sobre la organización de computadores.

1.2 OBJETIVOS

1.2.1 Objetivo general

El presente informe tiene como objetivo general el análisis comparativo de las técnicas para evitar hazards, utilizando una implementación de un simulador de pipeline para la detección de estos.

1.2.2 Objetivos específicos

Dentro de los objetivos para este informe están:

1. Generar un simulador básico del pipeline de cinco etapas de un procesador con un subconjunto de registros e instrucciones del *Instruction Set Architecture* de MIPS, para detectar hazards.
2. Aplicar técnicas de detección y de solución de hazards.
3. Fase de experimentación, se aplica la detección para cada técnica.
4. Análisis de los resultados, comparación de resultados entre técnicas.

1.3 PROBLEMA

El problema presentado en este informe consiste en que las diferentes técnicas utilizadas para evitar los hazards de control y datos dentro de un procesador, son más o menos efectivas dependiendo de distintos factores. Por lo cuál se hace necesario la detección de estos, así que se plantea la siguiente pregunta:

¿Qué técnica de hazard improvement al ser aplicada produce los mejores resultados?

Este informe presenta el planteamiento y solución a esta pregunta.

1.4 ORGANIZACIÓN DEL DOCUMENTO

El lector puede acceder a este documento y guiarse a través de los capítulos siguientes:

1. Introducción: la sección actual da al lector la información necesaria de preámbulo al tema en cuestión, motivaciones y objetivos esperados.
2. Marco Teórico: en esta sección se hará un resumen explicativo de conceptos que el lector necesita entender en cierta medida para la mejor comprensión del fondo de este informe.
3. Desarrollo: en esta sección el lector podrá conocer en detalle el proceso de construcción de este laboratorio.
4. Análisis: el lector tendrá acceso a los razonamientos obtenidos a través de la realización de experimentos sobre las distintas técnicas aplicadas a los archivos de prueba y su comparación.
5. Para finalizar existe una sección de conclusiones para el informe, acerca del trabajo realizado y de la comparación entre objetivos y logros alcanzados.

1.5 HERRAMIENTAS

Para la realización de este laboratorio se utilizaron las siguiente herramientas:

- Linux Mint 18 “Sarah” [1]
- Sublime Build 3126[2]
- Python 2.7.12 [3]

CAPÍTULO 2. MARCO TEÓRICO

A continuación se explican algunos conceptos que el lector debe entender para abordar la lectura de este informe de análisis.[4]

- ISA: Esta sigla corresponde a *Instructions Set Architecture*, es decir un conjunto conformado por aquellas instrucciones que soporta la arquitectura de un procesador.
- MIPS: Esta sigla corresponde a *Microprocessor without Interlocked Pipeline Stages*, es decir es un tipo de microprocesador en el cuál las instrucciones de su ISA se ejecutan de manera secuencial. y no hay interconexión entre las etapas.
- Pipeline: Este concepto se refiere a la distribución y funcionamiento de las etapas de un procesador, en el caso de este informe, se utiliza un pipeline de cinco etapas
- Registro: Este concepto se refiere a espacios de memoria dentro de los procesadores en donde se guardan temporalmente valores usados.
- Dependencia: En un programa escrito para MIPS así como también para otros procesadores, se producen las llamadas dependencias, las cuales suceden porque se utilizan los mismos registros para instrucciones consecutivas o cercanas en el código
- Hazards: Cuando existen dependencias pueden existir hazards, es decir eventos o amenazas a la integridad de las instrucciones que son ejecutadas en un programa, existen diferentes tipos, para efectos de este laboratorio se enfocan dos.
 - Hazards de datos: Este tipo de hazard hace referencia una amenaza a la integridad de los datos y el resultado final de un programa ya que existen dependencias con los registros utilizados en algunas de sus instrucciones. Por ejemplo, si una instrucción requiere un registro en la primera etapa de ID para una instrucción `add $t1,$t0,$t0` y previamente otra instrucción utiliza el mismo registro para guardar su resultado en él, p.e. `sub $t0,$t8,$t8`, entonces existe un hazard de datos dado que según el pipeline, las instrucciones sobrelapadas o paralelizadas provocarían que el registro utilizado en `sub $t0` no contendría el verdadero resultado necesario para que la siguiente instrucción `add` realice la operación correctamente.
 - Hazards de control: Este tipo de hazard hace referencia a una amenaza a la integridad de los datos y el resultado final de un programa debido a la utilización de instrucciones de control, es decir, instrucciones de `branch`, las cuales utilizan registros que están siendo utilizados por instrucciones anteriores que se encuentran en ejecución al llegar al `branch`.

CAPÍTULO 3. DESARROLLO

3.1 CONSTRUCCIÓN DE SIMULADOR

El simulador generado en esta etapa del laboratorio está basado en el pipeline de cinco etapa de los micropocesadores RISC correspondiente a MIPS, en el cuál se cuenta con las siguientes etapas:

- Instruction Fetch (IF)
- Instruction Decode (ID)
- Execution (EX)
- Memory (MEM)
- Write Back (WB)

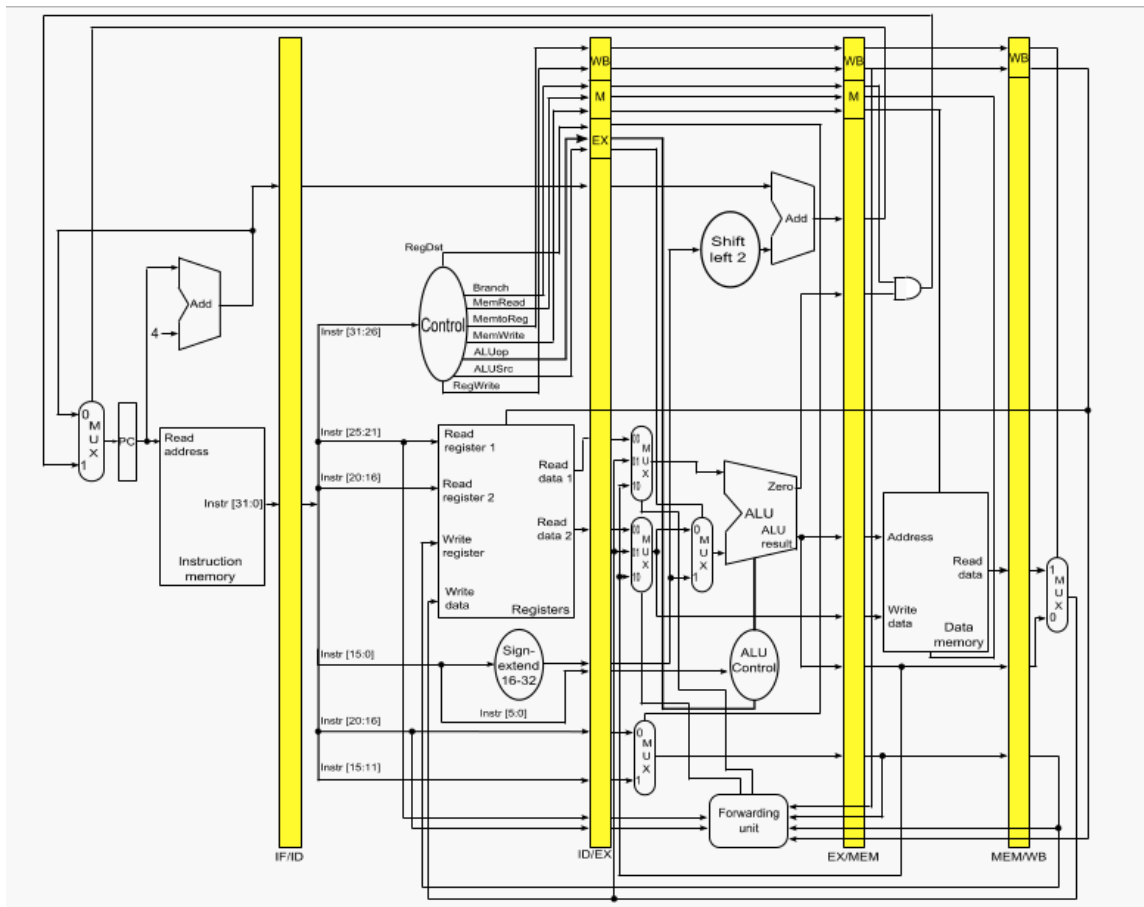


Figura 3-1: Pipeline de cinco etapas, base de la implementación[4]

Entre cada una de estas etapas existen buffers, es decir, existen espacios para almacenar datos entre etapas que estan siendo ejecutadas. A continuación se enlistan los buffers disponibles.

- IF/ID
- ID/EX
- EX/MEM
- MEM/WB

La Figura 3-1 corresponde a la distribución del pipeline que se utiliza para implementar la simulación, en donde es posible distinguir como rectángulos amarillos los buffers con sus respectivas líneas de entrada y salida. Además se construye cada uno de los componentes del datapath, como la memoria de instrucciones, la ALU, el conjunto de registros, la memoria, los multiplexores para las decisiones, y los buffers.

Dada la entrada proporcionada correspondiente a un programa con instrucciones de Para la memoria de instrucciones se cargan las instrucciones del archivo de entrada en un arreglo unidimensional sin los espacios vacios ni las etiquetas y para instrucciones de tipo I con etiquetas estas se cambian por los valores de los índices a los cuales les corresponde saltar.

Para la implementación de los componentes se utilizan en su mayoría, diccionarios otorgados como herramienta del lenguaje Python, con los cuales se representan las lineas de control y los datos que se almacenan. Estas estructuras funionan a modo de variable global, dado que sus valores deben permitir acceso desde todas las etapas.

Se tiene un contador global PC, representando al Program Counter con el cual se abordan las instrucciones de la instruction memory.

Los registros son inicializados con un valor cero tal y como se dispone en el simulador MARS utilizado en el curso y es conocido.

La principal estrategia de construcción de este simulador en base al pipeline es la utilización de la codificación de las instrucciones A continuación se explica etapa por etapa la estrategia de construcción.

3.1.1 Instruction Fetch IF

Para la etapa IF se obtiene la instruccion IR en el índice correspondiente la cual se codifica segun su tipo en un palabras de "hasta" 32 bits, se dice "hasta" ya que para efectos de cálculo posteriores dentro de la ALU algunas instrucciones de tipo I que pasan ignoran la Extensión de Signo del immediate agregando a la palabra solamente el numero entero.

Ejemplo:

$$addi \quad \$a0, \$a1, 4 \quad = \quad 001000 \quad 00100 \quad 00011 \quad 4 \quad (3.1)$$

En 3.1 se puede observar la codificación de una instrucción addi de tipo I, en el simulador construido. Por otra parte se aumenta el PC según el estado del multiplexor del salto en la etapa de MEM para el Branch. Se almacenan PC+4 y la instrucción codificada en el buffer IF/ID.

3.1.2 Instruction Decode ID

En esta etapa se actualiza la unidad de control de acuerdo a la instrucción proveniente del buffer IF/ID, se actualizan los registros, haciendo el fetch de los códigos de la instrucción, así como los datos se cargan desde el archivo de registro. Y finalmente se actualizan los datos del siguiente buffer ID/EX con los registros y datos obtenidos, además de las líneas de control seteadas.

3.1.3 Execution EX

En esta etapa se obtienen los datos del buffer previo y se operan en la ALU. Es clave reconocer en esta etapa los hazards de datos dentro del pipeline, una de las razones de la elección de esta forma de implementación del simulador, es que se pueden obtener los datos y entender de una manera más visual donde detectar los hazards de datos. Posterior a la operación en ALU y la revisión de hazards se actualiza el siguiente buffer EX/MEM con los datos resultantes.

3.1.4 Memory MEM

Ya en MEM dependiendo de la instrucción cargada en el buffer EX/MEM, se procede a acceder a memoria para escribir o leer, o bien no realizar acciones. Una vez terminado el proceso de acceso a memoria de datos, se actualiza el último buffer de MEM/WB donde se almacenan las últimas dos líneas de control Reg-Write y MemtoReg, además del nombre del registro en que se escriben los datos para la instrucción que se está ejecutando.

3.1.5 Write Back WB

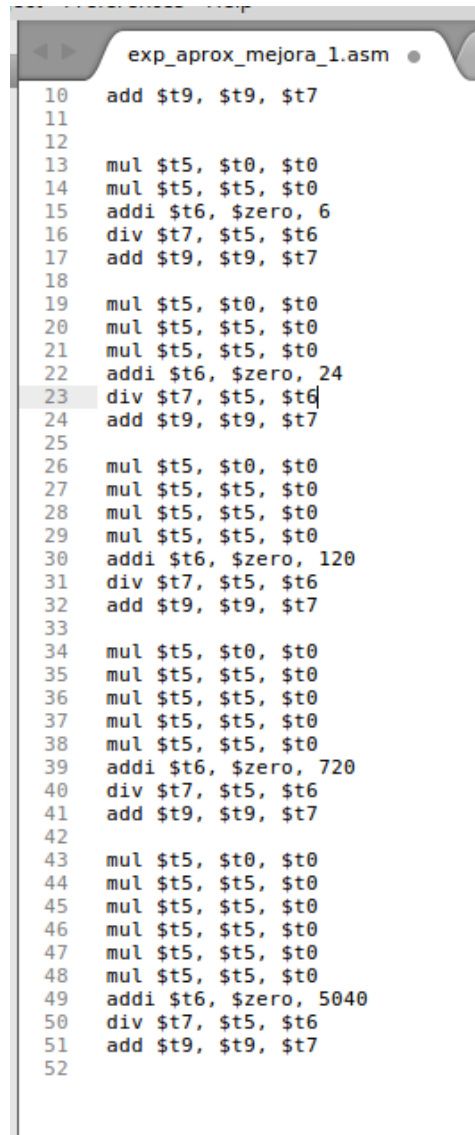
Con los datos del buffer EX/MEM se termina de ejecutar la instrucción seteando las últimas líneas y cambiando los datos en el registro.

Cada una de las etapas antes mencionadas son tratadas como funciones trabajando sobre variables globales y para ejecutar las etapas se introducen en una función de ejecución que las sobrepone unas con otras para que el funcionamiento respete el pipeline estudiado en clases.

3.2 RESULTADOS

A continuación se presentan los resultados obtenidos al utilizar el simulador con los programas de pruebas otorgado: `expr_aprox`, `suma_arreglo`, `factorial`, `fizz_buzz` y `fizz_buzz_div_2`. Es necesario destacar desde un inicio que solo el programa `expr_aprox` funciona en su totalidad, los demás programas presentan errores en la ejecución que no fueron solucionados.

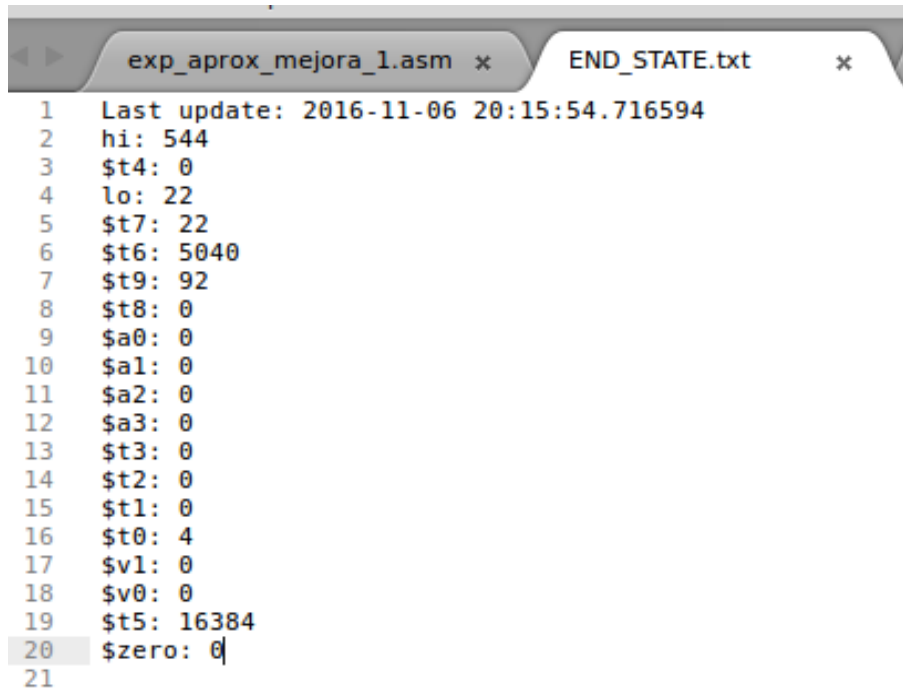
3.2.1 Expr_aprox



```
exp_aprox_mejora_1.asm
10  add $t9, $t9, $t7
11
12
13  mul $t5, $t0, $t0
14  mul $t5, $t5, $t0
15  addi $t6, $zero, 6
16  div $t7, $t5, $t6
17  add $t9, $t9, $t7
18
19  mul $t5, $t0, $t0
20  mul $t5, $t5, $t0
21  mul $t5, $t5, $t0
22  addi $t6, $zero, 24
23  div $t7, $t5, $t6
24  add $t9, $t9, $t7
25
26  mul $t5, $t0, $t0
27  mul $t5, $t5, $t0
28  mul $t5, $t5, $t0
29  mul $t5, $t5, $t0
30  addi $t6, $zero, 120
31  div $t7, $t5, $t6
32  add $t9, $t9, $t7
33
34  mul $t5, $t0, $t0
35  mul $t5, $t5, $t0
36  mul $t5, $t5, $t0
37  mul $t5, $t5, $t0
38  mul $t5, $t5, $t0
39  addi $t6, $zero, 720
40  div $t7, $t5, $t6
41  add $t9, $t9, $t7
42
43  mul $t5, $t0, $t0
44  mul $t5, $t5, $t0
45  mul $t5, $t5, $t0
46  mul $t5, $t5, $t0
47  mul $t5, $t5, $t0
48  mul $t5, $t5, $t0
49  addi $t6, $zero, 5040
50  div $t7, $t5, $t6
51  add $t9, $t9, $t7
52
```

Figura 3-2: Archivo de prueba *expr_aprox*

Esta ejecución se completa totalmente acabando con la ejecución de todas las instrucciones, obteniendo el archivo de END_STATE y HAZARDS, como es posible verlo en las Figura 3-3 y Figura 3-4. Los valores obtenidos en esta ejecución son los finales sin ninguna mejora realizada.

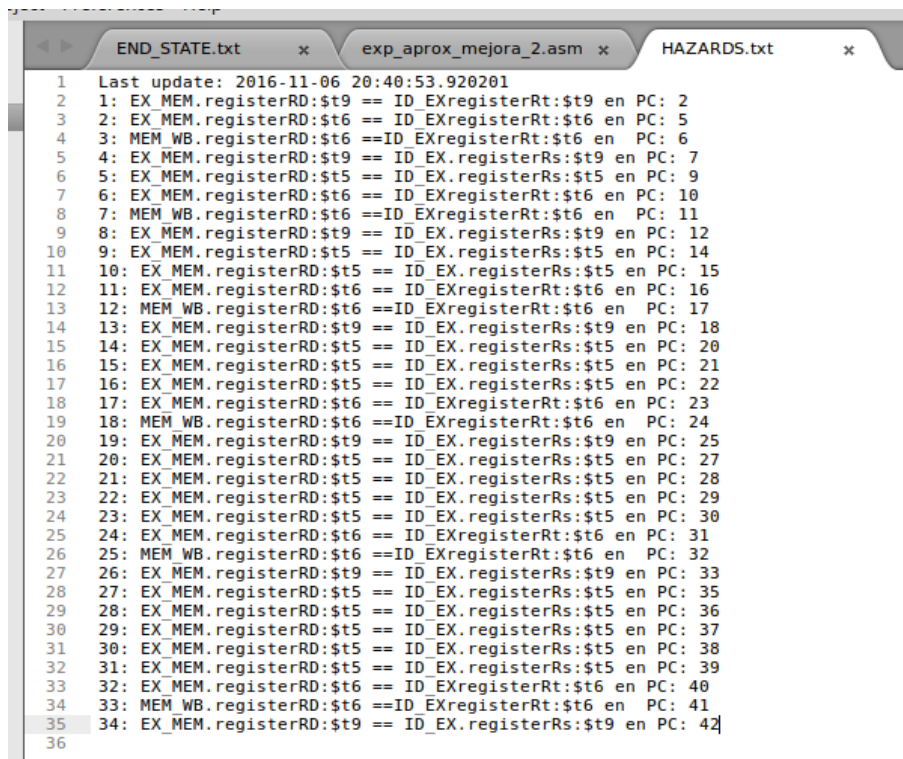


```

exp_aprox_mejora_1.asm x  END_STATE.txt x
1 Last update: 2016-11-06 20:15:54.716594
2 hi: 544
3 $t4: 0
4 lo: 22
5 $t7: 22
6 $t6: 5040
7 $t9: 92
8 $t8: 0
9 $a0: 0
10 $a1: 0
11 $a2: 0
12 $a3: 0
13 $t3: 0
14 $t2: 0
15 $t1: 0
16 $t0: 4
17 $v1: 0
18 $v0: 0
19 $t5: 16384
20 $zero: 0
21

```

Figura 3-3: END_STATE de expr_aprox

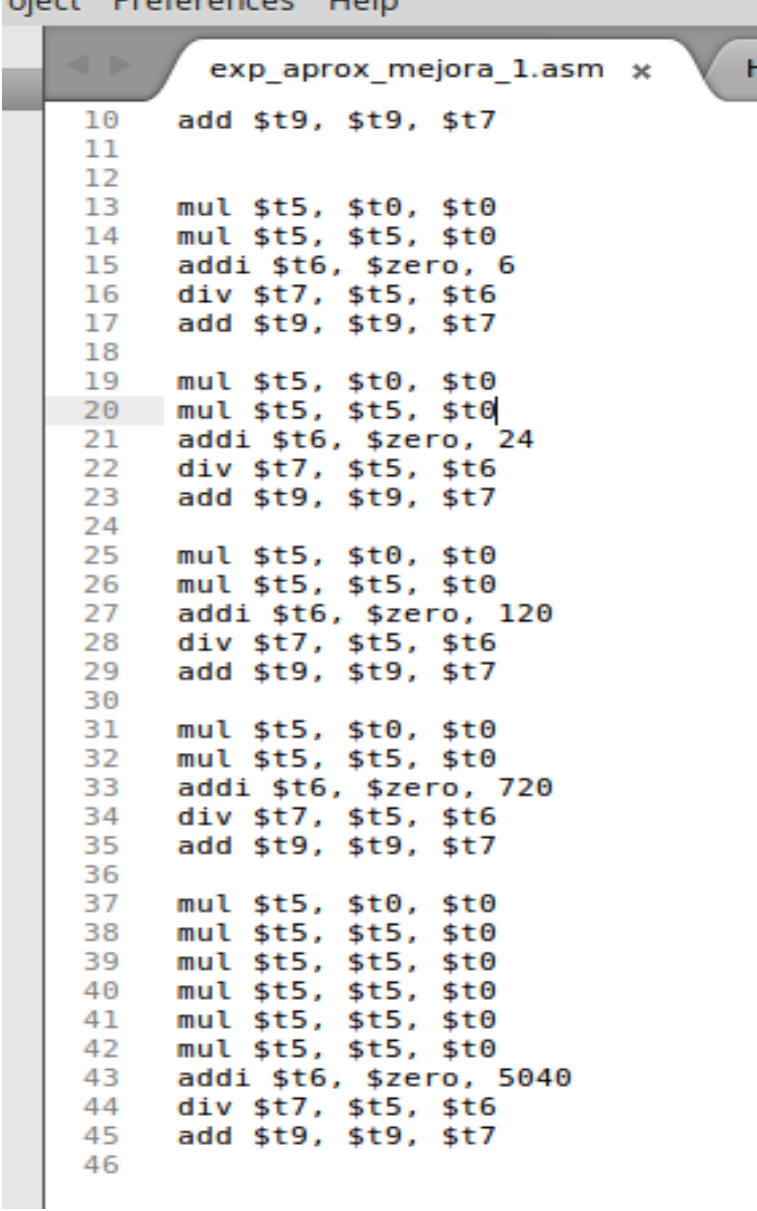


```

END_STATE.txt x  exp_aprox_mejora_2.asm x  HAZARDS.txt x
1 Last update: 2016-11-06 20:40:53.920201
2 1: EX_MEM.registerRD:$t9 == ID_EX.registerRt:$t9 en PC: 2
3 2: EX_MEM.registerRD:$t6 == ID_EX.registerRt:$t6 en PC: 5
4 3: MEM_WB.registerRD:$t6 ==ID_EX.registerRt:$t6 en PC: 6
5 4: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 7
6 5: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 9
7 6: EX_MEM.registerRD:$t6 == ID_EX.registerRt:$t6 en PC: 10
8 7: MEM_WB.registerRD:$t6 ==ID_EX.registerRt:$t6 en PC: 11
9 8: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 12
10 9: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 14
11 10: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 15
12 11: EX_MEM.registerRD:$t6 == ID_EX.registerRt:$t6 en PC: 16
13 12: MEM_WB.registerRD:$t6 ==ID_EX.registerRt:$t6 en PC: 17
14 13: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 18
15 14: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 20
16 15: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 21
17 16: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 22
18 17: EX_MEM.registerRD:$t6 == ID_EX.registerRt:$t6 en PC: 23
19 18: MEM_WB.registerRD:$t6 ==ID_EX.registerRt:$t6 en PC: 24
20 19: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 25
21 20: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 27
22 21: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 28
23 22: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 29
24 23: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 30
25 24: EX_MEM.registerRD:$t6 == ID_EX.registerRt:$t6 en PC: 31
26 25: MEM_WB.registerRD:$t6 ==ID_EX.registerRt:$t6 en PC: 32
27 26: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 33
28 27: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 35
29 28: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 36
30 29: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 37
31 30: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 38
32 31: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 39
33 32: EX_MEM.registerRD:$t6 == ID_EX.registerRt:$t6 en PC: 40
34 33: MEM_WB.registerRD:$t6 ==ID_EX.registerRt:$t6 en PC: 41
35 34: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 42
36

```

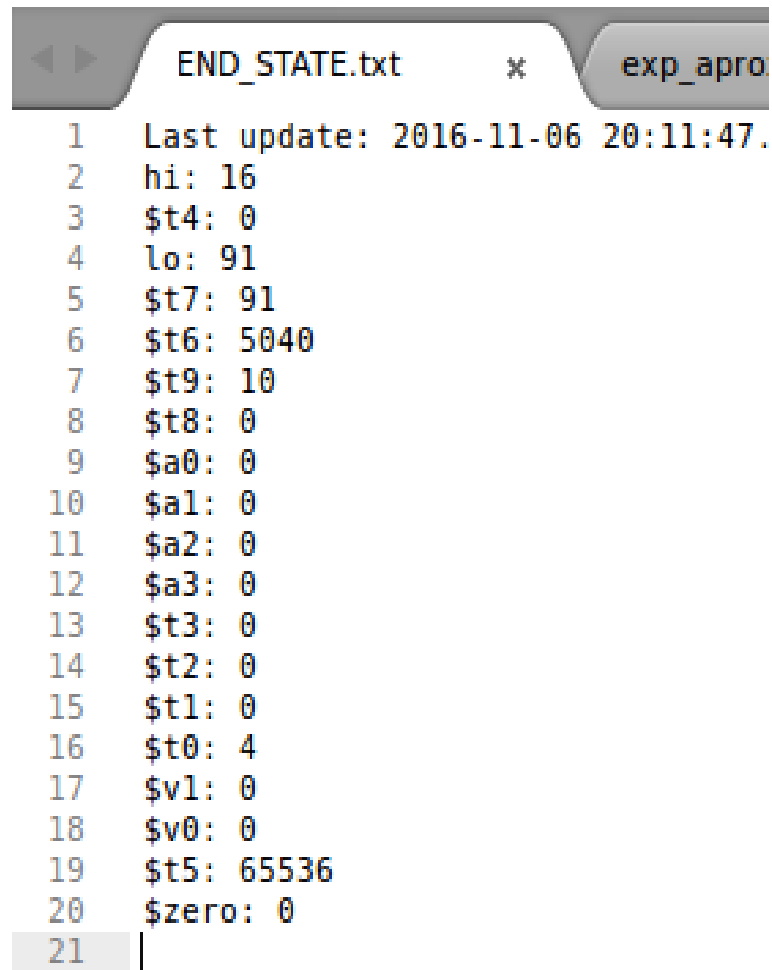
Figura 3-4: HAZARDS de expr_aprox

Mejora 1

```
10  add $t9, $t9, $t7
11
12
13  mul $t5, $t0, $t0
14  mul $t5, $t5, $t0
15  addi $t6, $zero, 6
16  div $t7, $t5, $t6
17  add $t9, $t9, $t7
18
19  mul $t5, $t0, $t0
20  mul $t5, $t5, $t0
21  addi $t6, $zero, 24
22  div $t7, $t5, $t6
23  add $t9, $t9, $t7
24
25  mul $t5, $t0, $t0
26  mul $t5, $t5, $t0
27  addi $t6, $zero, 120
28  div $t7, $t5, $t6
29  add $t9, $t9, $t7
30
31  mul $t5, $t0, $t0
32  mul $t5, $t5, $t0
33  addi $t6, $zero, 720
34  div $t7, $t5, $t6
35  add $t9, $t9, $t7
36
37  mul $t5, $t0, $t0
38  mul $t5, $t5, $t0
39  mul $t5, $t5, $t0
40  mul $t5, $t5, $t0
41  mul $t5, $t5, $t0
42  mul $t5, $t5, $t0
43  addi $t6, $zero, 5040
44  div $t7, $t5, $t6
45  add $t9, $t9, $t7
46
```

Figura 3-5: expr_aprox con mejora 1

En esta mejora se eliminaron líneas repetidas como por ejemplo las 28 y 29 de la Figura 3-2



```
1 Last update: 2016-11-06 20:11:47.
2 hi: 16
3 $t4: 0
4 lo: 91
5 $t7: 91
6 $t6: 5040
7 $t9: 10
8 $t8: 0
9 $a0: 0
10 $a1: 0
11 $a2: 0
12 $a3: 0
13 $t3: 0
14 $t2: 0
15 $t1: 0
16 $t0: 4
17 $v1: 0
18 $v0: 0
19 $t5: 65536
20 $zero: 0
21 |
```

Figura 3-6: *END_STATE* de *expr_aprox* mejora 1

```

object Preferences Help
exp_aprox_mejora_1.asm x HAZARDS.txt x
1 Last update: 2016-11-06 20:11:47.559503
2 1: EX_MEM.registerRD:$t9 == ID_EXregisterRt:$t9 en PC: 2
3 2: EX_MEM.registerRD:$t6 == ID_EXregisterRt:$t6 en PC: 5
4 3: MEM_WB.registerRD:$t6 ==ID_EXregisterRt:$t6 en PC: 6
5 4: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 7
6 5: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 9
7 6: EX_MEM.registerRD:$t6 == ID_EXregisterRt:$t6 en PC: 10
8 7: MEM_WB.registerRD:$t6 ==ID_EXregisterRt:$t6 en PC: 11
9 8: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 12
10 9: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 14
11 10: EX_MEM.registerRD:$t6 == ID_EXregisterRt:$t6 en PC: 15
12 11: MEM_WB.registerRD:$t6 ==ID_EXregisterRt:$t6 en PC: 16
13 12: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 17
14 13: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 19
15 14: EX_MEM.registerRD:$t6 == ID_EXregisterRt:$t6 en PC: 20
16 15: MEM_WB.registerRD:$t6 ==ID_EXregisterRt:$t6 en PC: 21
17 16: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 22
18 17: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 24
19 18: EX_MEM.registerRD:$t6 == ID_EXregisterRt:$t6 en PC: 25
20 19: MEM_WB.registerRD:$t6 ==ID_EXregisterRt:$t6 en PC: 26
21 20: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 27
22 21: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 29
23 22: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 30
24 23: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 31
25 24: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 32
26 25: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 33
27 26: EX_MEM.registerRD:$t6 == ID_EXregisterRt:$t6 en PC: 34
28 27: MEM_WB.registerRD:$t6 ==ID_EXregisterRt:$t6 en PC: 35
29 28: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 36
30

```

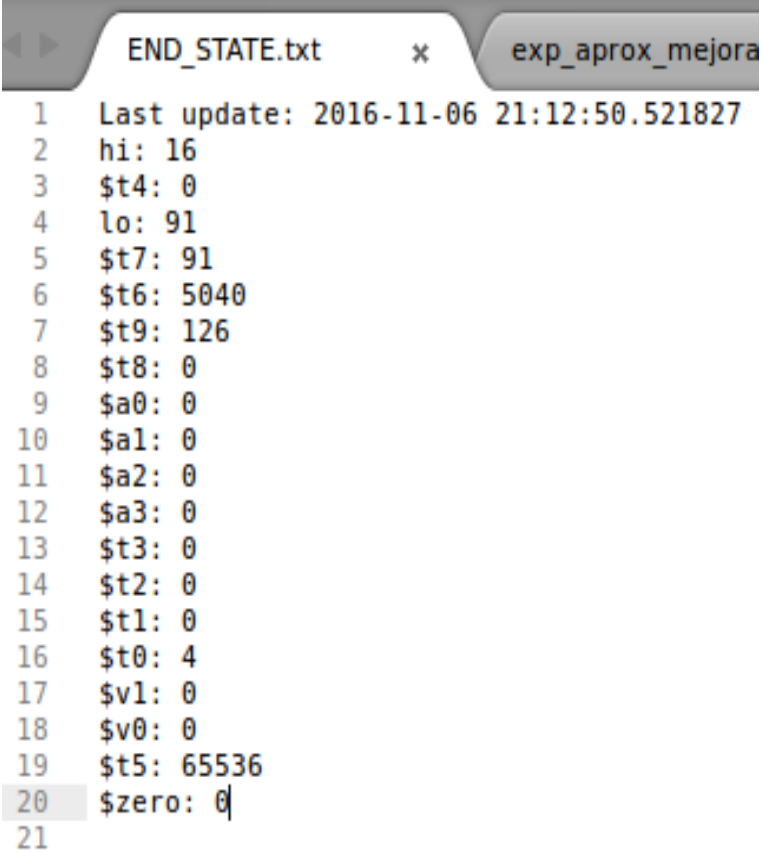
Figura 3-7: HAZARDS de *expr_aprox mejora 1*

Mejora 2

```
14    mul $t5, $t5, $t0
15    addi $t6, $zero, 6
16    div $t7, $t5, $t6
17    add $t9, $t9, $t7
18
19    mul $t5, $t0, $t0
20    add $t3, $zero, $zero
21    add $t3, $zero, $zero
22    add $t3, $zero, $zero
23    mul $t5, $t5, $t0
24    addi $t6, $zero, 24
25    div $t7, $t5, $t6
26    add $t9, $t9, $t7
27
28    mul $t5, $t0, $t0
29    add $t3, $zero, $zero
30    add $t3, $zero, $zero
31    add $t3, $zero, $zero
32    mul $t5, $t5, $t0
33    addi $t6, $zero, 120
34    div $t7, $t5, $t6
35    add $t9, $t9, $t7
36
37    mul $t5, $t0, $t0
38    add $t3, $zero, $zero
39    add $t3, $zero, $zero
40    add $t3, $zero, $zero
41    mul $t5, $t5, $t0
42    addi $t6, $zero, 720
43    div $t7, $t5, $t6
44    add $t9, $t9, $t7
```

Figura 3-8: expr_aprox con mejora 2

Se le agregan operaciones de suma de ceros, para eliminar dependencias entre las instrucciones de multiplicación, como es posible distinguir en Figura 3-8 y comparando con el archivo original de prueba en Figura 3-2



The image shows a terminal window with two tabs: 'END_STATE.txt' and 'exp_aprox_mejora'. The 'END_STATE.txt' tab is active, displaying a list of 21 lines of text. The text is as follows:

```
1 Last update: 2016-11-06 21:12:50.521827
2 hi: 16
3 $t4: 0
4 lo: 91
5 $t7: 91
6 $t6: 5040
7 $t9: 126
8 $t8: 0
9 $a0: 0
10 $a1: 0
11 $a2: 0
12 $a3: 0
13 $t3: 0
14 $t2: 0
15 $t1: 0
16 $t0: 4
17 $v1: 0
18 $v0: 0
19 $t5: 65536
20 $zero: 0
21
```

Figura 3-9: END_STATE de expr_aprox mejora 2

```

1 Last update: 2016-11-06 21:12:50.521945
2 1: EX_MEM.registerRD:$t9 == ID_EXregisterRt:$t9 en PC: 2
3 2: EX_MEM.registerRD:$t6 == ID_EXregisterRt:$t6 en PC: 5
4 3: MEM_WB.registerRD:$t6 ==ID_EXregisterRt:$t6 en PC: 6
5 4: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 7
6 5: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 9
7 6: EX_MEM.registerRD:$t6 == ID_EXregisterRt:$t6 en PC: 10
8 7: MEM_WB.registerRD:$t6 ==ID_EXregisterRt:$t6 en PC: 11
9 8: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 12
10 9: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 14
11 10: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 15
12 11: EX_MEM.registerRD:$t6 == ID_EXregisterRt:$t6 en PC: 16
13 12: MEM_WB.registerRD:$t6 ==ID_EXregisterRt:$t6 en PC: 17
14 13: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 18
15 14: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 20
16 15: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 21
17 16: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 22
18 17: EX_MEM.registerRD:$t6 == ID_EXregisterRt:$t6 en PC: 23
19 18: MEM_WB.registerRD:$t6 ==ID_EXregisterRt:$t6 en PC: 24
20 19: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 25
21 20: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 30
22 21: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 34
23 22: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 38
24 23: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 42
25 24: EX_MEM.registerRD:$t6 == ID_EXregisterRt:$t6 en PC: 46
26 25: MEM_WB.registerRD:$t6 ==ID_EXregisterRt:$t6 en PC: 47
27 26: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 48
28 27: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 50
29 28: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 51
30 29: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 52
31 30: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 53
32 31: EX_MEM.registerRD:$t5 == ID_EX.registerRs:$t5 en PC: 54
33 32: EX_MEM.registerRD:$t6 == ID_EXregisterRt:$t6 en PC: 55
34 33: MEM_WB.registerRD:$t6 ==ID_EXregisterRt:$t6 en PC: 56
35 34: EX_MEM.registerRD:$t9 == ID_EX.registerRs:$t9 en PC: 57
36

```

Figura 3-10: HAZARDS de expr_aprox mejora 2

3.2.2 Factorial

En este archivo al ejecutarse se genera un error, pero la actualización parcial de los archivos de estado de registros y de hazards estan disponibles a pesar del error para obtener los resultados parciales. En la Figura 3-11 es visible que se alcanzan a detectar algunos de los posibles hazards del archivo de prueba parcialmente ejecutado.

```

1 Last update: 2016-11-06 21:34:38.977843
2 1: EX_MEM.registerRD:$t1 == ID_EXregisterRt:$t1 en PC: 2
3 2: EX_MEM.registerRD:$t2 == ID_EXregisterRt:$t2 en PC: 3
4 3: EX_MEM.registerRD:$t0 == ID_EXregisterRt:$t0 en PC: 4
5 4: EX_MEM.registerRD:$t0 == ID_EX.registerRs:$t0 en PC: 6
6

```

Figura 3-11: HAZARDS de factorial sin mejora

3.2.3 Suma_arreglo

No hay ejecución debido a error con instrucción la y la dirección de ARREGLO.

3.2.4 Fizz.buzz

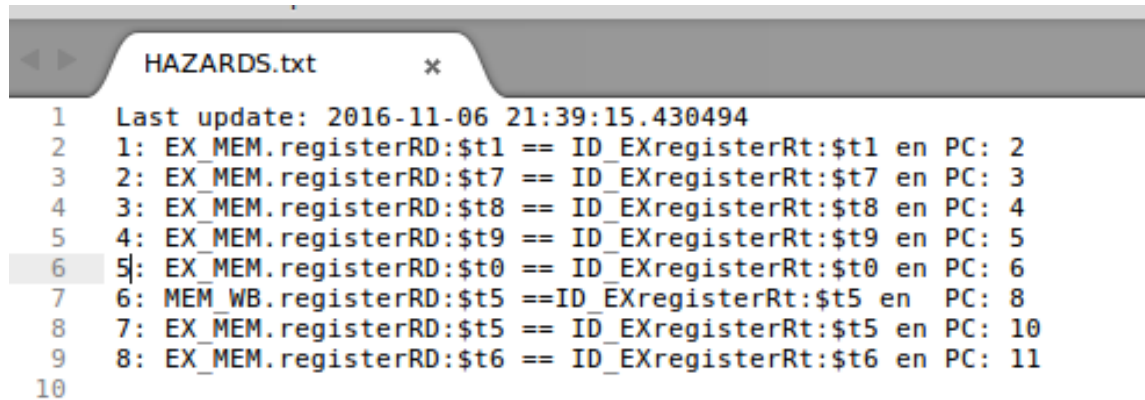
Este archivo se ejecuta, pero queda dentro de la condición de termino de la ejecución, donde PC no alcanza el final de la instruction memory, porque una de las instrucciones no ejecuta y no cambia el valor del contador del programa.

```

1 Last update: 2016-11-06 21:39:15.430903
2 hi: 0
3 $t4: 0
4 lo: 0
5 $t7: 0
6 $t6: 0
7 $t9: 0
8 $t8: 0
9 $a0: 0
10 $a1: 0
11 $a2: 0
12 $a3: 0
13 $t3: 0
14 $t2: 0
15 $t1: 100
16 $t0: 0
17 $v1: 0
18 $v0: 0
19 $t5: 15
20 $zero: 0
21

```

Figura 3-12: END_STATE de fizz.buzz sin mejora



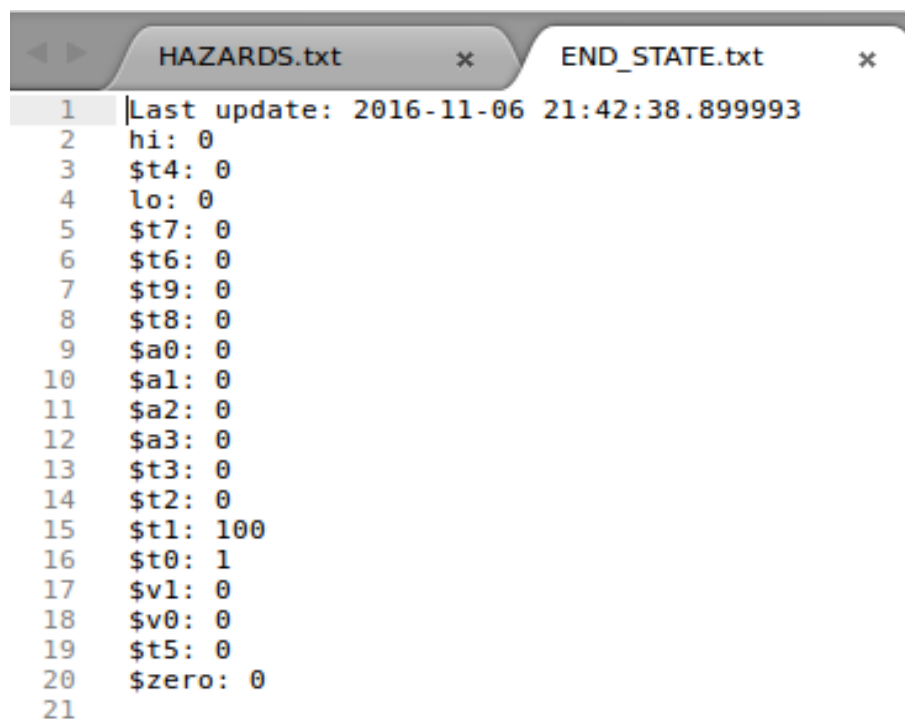
```

1 Last update: 2016-11-06 21:39:15.430494
2 1: EX_MEM.registerRD:$t1 == ID_EXregisterRt:$t1 en PC: 2
3 2: EX_MEM.registerRD:$t7 == ID_EXregisterRt:$t7 en PC: 3
4 3: EX_MEM.registerRD:$t8 == ID_EXregisterRt:$t8 en PC: 4
5 4: EX_MEM.registerRD:$t9 == ID_EXregisterRt:$t9 en PC: 5
6 5: EX_MEM.registerRD:$t0 == ID_EXregisterRt:$t0 en PC: 6
7 6: MEM_WB.registerRD:$t5 ==ID_EXregisterRt:$t5 en PC: 8
8 7: EX_MEM.registerRD:$t5 == ID_EXregisterRt:$t5 en PC: 10
9 8: EX_MEM.registerRD:$t6 == ID_EXregisterRt:$t6 en PC: 11
10

```

Figura 3-13: HAZARDS de fizz_buzz sin mejora

3.2.5 Fizz_buzz_div_2



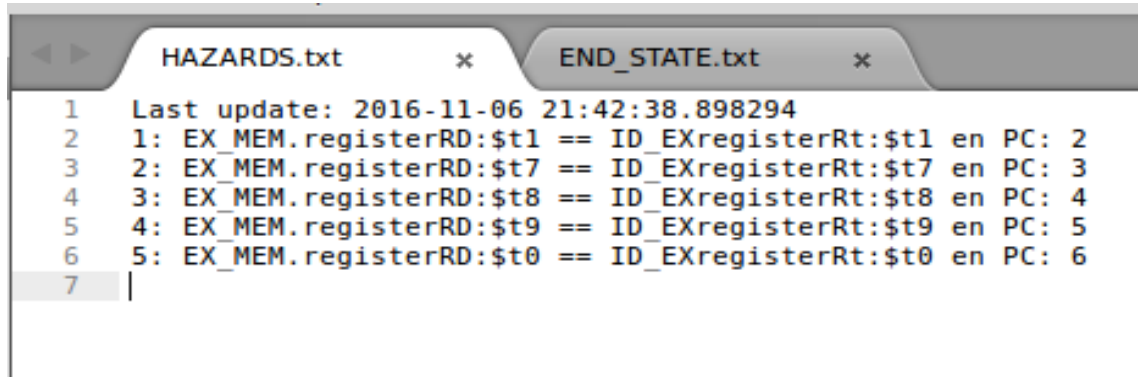
```

1 Last update: 2016-11-06 21:42:38.899993
2 hi: 0
3 $t4: 0
4 lo: 0
5 $t7: 0
6 $t6: 0
7 $t9: 0
8 $t8: 0
9 $a0: 0
10 $a1: 0
11 $a2: 0
12 $a3: 0
13 $t3: 0
14 $t2: 0
15 $t1: 100
16 $t0: 1
17 $v1: 0
18 $v0: 0
19 $t5: 0
20 $zero: 0
21

```

Figura 3-14: END_STATE de fizz_buzz_div_2 sin mejora

La ejecución de este archivo de prueba termina debido a un error de tipo con un elemento no inicializado en el programa. Es posible rescatar los resultados parciales en la Figura 3-14 y Figura 3-15



```
1 Last update: 2016-11-06 21:42:38.898294
2 1: EX_MEM.registerRD:$t1 == ID_EXregisterRt:$t1 en PC: 2
3 2: EX_MEM.registerRD:$t7 == ID_EXregisterRt:$t7 en PC: 3
4 3: EX_MEM.registerRD:$t8 == ID_EXregisterRt:$t8 en PC: 4
5 4: EX_MEM.registerRD:$t9 == ID_EXregisterRt:$t9 en PC: 5
6 5: EX_MEM.registerRD:$t0 == ID_EXregisterRt:$t0 en PC: 6
7 |
```

Figura 3-15: HAZARDS de *fizz_buzz_div_2* sin mejora

CAPÍTULO 4. ANÁLISIS

4.0.1 Expr_aprox

Mejora 1

Se obtuvieron los mismos resultados finales para el programa según Figura 3-6 y Figura 3-3. Esto se debe a que al eliminar líneas repetidas la ejecución de estas solo aumentan la cantidad de instrucciones ejecutadas pero la tarea sigue siendo la misma.

Se observa en la Figura 3-7 que al compararlos con los de la figura Figura 3-4 se disminuyen debido a la mejora.

Mejora 2

En el caso de la mejora dos para este archivo, se observa que no es posible reducir los hazards en base al uso de esta mejora, según Figura 3-10 se observan más hazards en comparación con el estado sin mejora.

4.0.2 Suma_arreglo

No se obtienen resultados suficientes para el análisis.

4.0.3 Factorial

No se obtienen resultados suficientes para el análisis.

4.0.4 Fizz_buzz

No se obtienen resultados suficientes para el análisis.

4.0.5 Fizz_buzz_div_2

No se obtienen resultados suficientes para el análisis.

CAPÍTULO 5. CONCLUSIÓN

Como medida de conclusión se procede a medir el nivel alcanzado del laboratorio con respecto a los objetivos descritos en la sección inicial.

Se determina que no fue posible construir el simulador del pipeline de manera efectiva y que cumpliera con todos los objetivos.

En definitiva se construyó un programa que resuelve instrucciones pero que falla en condiciones de branch y que no distingue los hazards de control, solo reconoce los hazards de datos, por lo tanto no cumple con lo establecido en cuanto a la detección de hazards.

En cuanto al análisis prometido, solo se puede establecer para uno de los archivos que otorga resultados y no así para todas las técnicas y mejoras esperadas.

En resumen, la implementación del pipeline con la codificación de la instrucción resultó en una complicación excesiva de la parte de este programador, el cual pudo implementar una solución parcial que no cumple con las expectativas que el lector buscaba de este informe de análisis.

CAPÍTULO 6. BIBLIOGRAFÍA

- [1] L. M. project, 2016. [Online]. Available: <https://www.linuxmint.com/>.
- [2] J. Skinner, 2008. [Online]. Available: <https://www.sublimetext.com/>.
- [3] G. van Rossum, [Online; accessed 1-October-2016], 1991. [Online]. Available: <https://www.python.org/>.
- [4] W. D. McQuain, [Online; accessed 30-October-2016], 2013. [Online]. Available: <http://courses.cs.vt.edu/cs2506/Spring2013/Notes/L11.PipelineControl.pdf>.
- [5] J. H. Gerry Kane, [Online; accessed 27-September-2016]. [Online]. Available: <https://www.d.umn.edu/~gshute/mips/single-cycle-summary.pdf>.
- [6] J. A. G. Pulido, *Repertorio de instrucciones mips*, [Online; accessed 27-September-2016], 2000. [Online]. Available: <https://www.d.umn.edu/~gshute/mips/single-cycle-summary.pdf>.