Implementació d'un entorn interactiu educatiu per a la docència d'Arquitectures RISC-V

Ayoub Hammani Abbasi i David Castells Rufas

Resum—Aquest treball se centra en el desenvolupament d'una interfície gràfica i un microprocessador RV32I microprogramat que formaran un entorn interactiu didàctic per a la docència d'arquitectures RISC-V. Aquest entorn interactiu té l'objectiu de proporcionar als estudiants una eina on poder experimentar i observar el funcionament i els components d'un processador, en aquest cas d'un microprocessador RV32I microprogramat. En el següent article, però, el model del microprocessador a desenvolupar diferirà de l'estructura òptima per la qual ha estat dissenyada l'ISA d'un processador RISC.

Paraules clau— Educació, Interfície gràfica, Microprocessadors, Python, RISC-V

Abstract— This paper is going to develop an interactive environment compound by a graphical interface and a microprogrammed microprocessor RV32I. The environment objective is to provide a tool to the students where they will be able to observe and interact with the components of a microprocessor RV32I. The structure of the microprocessor model that is going to be implemented will differ from the optimal structure for which was thought the RISC-V ISA.

Index Terms— Education, Graphical interface, Microprocessors, Python, RISC-V

1 INTRODUCCIÓ

Actualment, en la formació d'estudiants d'enginyeria informàtica s'ha detectat una lleugera mancança de comprensió sobre el funcionament dels processadors. L'aprenentatge comença des de ben aviat, concretament en el primer any del grau, amb l'assignatura Fonaments dels Computadors, on s'explica la seva estructura i programació a base de textos, pseudocodi i diagrames. Això permet a l'estudiant obtenir una primera idea dels components que els formen i el seu funcionament, no obstant això aquests coneixements no s'acaben d'assolir. Arribats aquí, ens podria sorgir la següent qüestió: quin és el vertader motiu d'aquest fenomen?

Partim del punt que l'estudiant és capaç d'adquirir una idea poc sòlida de l'estructura i funcionament bàsic d'un computador. L'enfocament més clar a seguir que s'observa es pot definir amb la pregunta següent: com aconseguir solidificar una mica més aquests conceptes? La resposta no és un altre que la pràctica.

D'altra banda, estem assistint ara mateix al naixement d'un nou processador, el RISC-V que, a més de fer una proposta d'ISA standard molt depurat, ofereix aquest ISA lliure i en obert (open-source). D'aquesta forma qualsevol empresa o particular pot fer-ne implementacions ja siguin comercials o públiques, sense haver de pagar cap mena de peatge per accedir a aquest ISA RISC-V [12].

El projecte va començar en 2010 a la Universitat de Califòrnia en Berkley. RISC-V es tracta d'un desenvolupament derivat de diversos projectes acadèmics de disseny de computadors, orientats a la investigació i la docència. Berkley s'inspira en les quatre generacions de projectes RISC liderades per David Patterson a l'any 1980, per definir el nom d'aquest nou risc com a "RISC cinc".

Per tant, observant el nou impuls que s'està generant al voltant d'aquesta "nova" arquitectura perquè pugui arribar al major nombre de desenvolupadors i empreses, s'ha considerat força beneficiós orientar el desenvolupament d'aquest entorn interactiu a través de l'arquitectura RISC-V. I d'aquesta manera contribuir a facilitar l'aprenentatge dels processadors als estudiants, alhora que es formen en una arquitectura emergent i en creixement que pot arribar a tenir un gran impacte en el futur.

2 OBJECTIUS

L'objectiu principal del següent TFG és la implementació d'un sistema didàctic per explicar el funcionament dels microprocessadors, concretament RISC-V. El sistema estarà compost principalment per una interfície gràfica i interactiva que permetrà explotar amb més facilitat l'entorn. Aquesta interfície serà connectada a un nucli

bàsic de RISC-V de 32 bits (RV32I) microprogramat.

Pel desenvolupament de la interfície s'ha definit les següents característiques o funcionalitats que haurà d'incorporar:

- ❏ Compilar un programa en llenguatge d'alt nivell (C/C++) i mostrar el codi resultat en asm.
- ❏ Executar pas a pas les instruccions asm..
- ❏ Executar pas a pas les instruccions de Microprograma.
- ❏ Visualitzar l'arquitectura del processador així com els valors a les entrades dels mòduls.
- ❏ Visualitzar el contingut de la memòria.

Un segon objectiu del treball va relacionat amb l'estratègia d'ensenyament del funcionament dels computadors basant-se en l'ISA, que origina en l'estudiant la falsa creença que un ISA només pot funcionar en l'arquitectura per la qual està dissenyat. Es considera que a efectes acadèmics resulta interessant demostrar la falsedat d'aquesta suposició, i s'implementarà un RISC-V microprogramat de 32 bits sense pipeline que diferirà de l'arquitectura ideada pel ISA d'un processador RISC.

3 ESTAT DE L'ART

Actualment, la interfície gràfica (o GUI) [9] és l'estàndard de disseny en la programació d'aplicacions de software destinades a usuaris. Permeten l'ús d'ordinadors i altres dispositius electrònics de forma intuïtiva mitjançant la manipulació directa d'elements gràfics com menús, finestres, botons, ratolins, entre altres.

Apareixen com a substituts del CLI, interfície de línia de comandes, que permet als usuaris comunicar-se amb la computadora mitjançant comandes de text per portar a terme tasques específiques. Perquè l'ordinador comprengui la comanda i porti a terme la tasca demanada, és necessari introduir el nom de la comanda exacte. Això produeix una dificultat afegida per a tots els nous usuaris d'ordinadors, resultant una interfície poc intuïtiva i natural.

El microprocessador [4] és la unitat central d'un sistema informàtic que realitza operacions lògiques i aritmètiques, que generalment inclouen sumar, restar, moure dades d'una posició a un altre i comparar valors. És un dispositiu programable i versàtil que incorpora les funcionalitats d'una CPU en un circuit integrat.

Un microprocessador rep dades en binari com a input, les processa, i després genera un output basat en les instruccions emmagatzemades en memòria. Les dades són processades per l'ALU (unitat aritmètica lògica), la CU (control unit), i un array de registres. L'array de registres actua com una espècie de memòria ràpida temporal durant el processament de les dades, per evitar l'accés a memòria continu. La CU s'encarrega de coordinar i controlar el flux de les dades i les instruccions pel sistema mitjançant senyals de control.

4 METODOLOGIA

4.1 INTERFÍCIE GRÀFICA

4.1.1 Llenguatge

S'ha escollit Python perquè compta amb una gran quantitat de frameworks pel desenvolupament de GUI's, des de Tkinter com a llibreria per defecte instal·lada fins a una important quantitat de solucions mulitplataforma, com PyQT o wxPython que poden ser instal·lades com a llibreries third-party[3].

És conegut com un llenguatge fàcil d'entendre, escriure i mantenir, compta amb moltes llibreries que permeten utilitzar funcionalitats de forma simple i efectiva. Programar la interfície amb aquest llenguatge, permetrà tenir accés a diferents llibreries d'implementació de GUIs, i produir un codi estructurat i comprensible que facilitarà tant el seu manteniment com la seva evolució amb noves funcionalitats.

4.1.2 Frameworks GUI

Un dels motius pel que s'ha escollit el llenguatge Python, ja esmentat, és la gran varietat de frameworks disponibles per implementar interfícies gràfiques. Entre la gran varietat, en destaquem els següents.

4.1.2.1 PyQt [10]

És un dels enllaços (bindings) multiplataforma Python més coneguts i utilitzats, implementat amb la llibreria Qt, que utilitza el framework de desenvolupament d'aplicacions Qt. Combina les millors funcionalitats de Python i Qt per oferir a l'usuari la possibilitat de crear interfícies gràfiques de dos modes diferents, programant en codi o visualment mitjançant el dissenyador gràfic Qt. Està disponible per la majoria d'entorns: Unix/Linux, Mac OS X, Sharp Zaurus i Windows.

4.1.2.2 Kivy [14]

Està escrit amb una barreja de Python i Cython, i és un framework de codi obert molt utilitzat en la implementació d'interfícies gràfiques en aplicacions multitàctils que fan servir la interfície d'usuari natural (NUI).

Kivy permet als dissenyadors codificar una sola vegada la interfície i poder desplegar-la en múltiples plataformes. L'ús més habitual d'aquest framework és en les aplicacions Android i iOS, però també es pot fer servir sense cap inconvenient en el desenvolupament de GUIs per Linux, Windows, Mac OS i Raspberry Pi.

4.2.3 Tkinter[11]

És la biblioteca per defecte en Python i permet als usuaris crear elements gràfics de forma senzilla que formaran en conjunt la interfície gràfica. En el següent treball la interfície gràfica ha estat implementada mitjançant aquesta llibreria.

Tkinter es compon de widgets que poden ser utilitzats per crear títols, camps de text, botons, estructures de dades, etc. I, una vegada són creats es podran enllaçar a diverses funcionalitats, mètodes, o widgets, i interactuar amb aquests, proporcionant una interfície intuïtiva i interactiva. Cadascun d'ells ofereix un nivell de personalització diferent.

Un dels avantatges de TKinter, i punt important a l'hora d'escollir-lo, és la quantitat d'informació i recursos disponibles sobre aquesta interfície gràfica. També compta amb una gran comunitat activa on l'usuari pot trobar una gran quantitat de persones que poden resoldre els seus dubtes.

Per tots els motius exposats, es va decidir utilitzar TKinter degut que ofereix una gran quantitat de funcionalitats per construir GUIs bàsiques, ve per defecte instal·lada en Python, i compta amb una varietat de recursos que permetrán consultar i resoldre els dubtes i errors que apareguin. A diferència d'algunes llibreries com PyQt, Tkinter no inclou un dissenyador gràfic per crear GUIs, de manera que totes les funcionalitats i elements gràfics s'hauran de programar a mà, on es podrà aprendre i desenvolupar una idea bàsica sobre el funcionament arrel de les interfícies gràfiques.

4.1.3 Widgets utilitzats

La interfície gràfica creada esta composta per múltiples widgets que proporcionen diferents utilitats que es descriuen a continuació.

El primer element gràfic que es pot apreciar al executar la GUI, és un widget anomenat "label" que permet imprimir per pantalla una sentència, i és utilitzat per definir els títols de cada component de dades (il·lustració 5).

Seguidament, trobem un widget de tipus text que permet escriure manualment diversos caràcters amb les funcionalitats bàsiques de qualsevol editor de text. És utilitzat per introduir el codi C/C++ que serà compilat i executat en el model de processador (il·lustració 6).

El següent element és un dels més comuns i utilitzats en la majoria de les GUI, el botó. Es poden observar múltiples botons en la interfície tal i com es pot apreciar en la il·lustració 6, i cada un d'aquests està enllaçat a una funcionalitat única i descrita en el seu nom. De forma que al prémer el botó s'executarà una rutina que gestionarà la tasca que ha de realitzar.

I l'últim widget que més s'ha fet servir ha estat el treeview (vista d'arbre), un element que permet presentar la informació de forma jeràrquica que en aquest cas, s'ha utilitzat per mostrar les dades en format columnes del codi assemblador (asm) i el valor dels registres del processador (il·lustració 8 i 10).

4.1.4 Fitxers ELF

ELF [13] és l'abreviatura de Format Executable i enllaçable (Executable and Linkable Format) i defineix l'estructura dels binaris, llibreries, i fitxers de nucli. L'especificació del format permet que el sistema operatiu interpreti correctament les instruccions màquina. Els fitxers ELF són, normalment, la sortida en format binari d'un compilador o enllaçable. En qualsevol sistema operatiu és necessari transformar les funcions que definim amb un llenguatge de programació d'alt nivell, al llenguatge que és capaç d'interpretar la CPU coneguda com a codi màquina. El procés comença amb el compilador que converteix aquestes funcions en codi d'objecte (ensamblador o bytecode). Aquest codi resultant és vinculat a un programa complet mitjançant una eina d'enllaçament. El resultat és un arxiu binari que ja pot ser executat en la CPU actual.

Els fitxers ELF es poden fer servir com arxius binaris, executables, codis d'objecte, i llibreries compartides. L'especificació del ELF també és utilitzada en el kernel de Linux i els seus mòduls.

Degut al disseny extensible del fitxer ELF, l'estructura difereix per cada fitxer. Però tots els fitxers ELF estan dividits en dues parts clares: la capçalera (header) i les dades (data).

4.1.4.1 ELF Header

És la part del fitxer on estan definides totes les característiques necessàries perquè les dades que conté es puguin interpretar correctament en el procés d'execució o vinculació. Tots els fitxers ELF han d'incloure una capçalera, i en són exemples del seu contingut camps com:

1. *Magic*: defineix els aspectes bàsics del fitxer (tipus ELF).
2. *Class*: específica l'arquitectura que ha generat el fitxer (32 o 64 bits).
3. *Data*: declara el format per interpretar les dades (little o big endian).
4. *OS/ABI*: esmenta el tipus i la versió de ABI que s'utilitza.
5. *Version*: versió del fitxer. Actualment només n'hi ha una.
6. *ABI version*: si es modifica la versió del ABI sortirà especificada aquí.
7. *Type*: aquest camp específica el propòsit del fitxer ELF. Els més comuns són executables i arxius binaris.
8. *Machine*: Informa del tipus de màquina on espera ser executat (exemple RV32I).

4.1.4.2 ELF Data

Aquesta és la part del fitxer on radiquen les dades, tal i com el seu nom indica, i en podem diferenciar dos components: segments i seccions.

Els segments contenen la informació necessària per portar a terme l'execució del fitxer. Un segment pot contenir zero o més seccions.

Les seccions en canvi, proporcionen la informació requerida alhora de portar a terme la vinculació (linking) i la recol·locació (relocation) de dades. Com s'ha comentat

amb anterioritat, els fitxers ELF poden diferir en el seu contingut de dades i per tant presentar diferents seccions, però principalment tots els fitxers executables presentaran com a mínim aquestes quatre:

- *.text*: conté el codi executable. Serà emmagatzemat en un segment amb accés de lectura i execució. Només es carrega una vegada i es manté en memòria.
- *.data*: està format per dades inicialitzades amb drets d'accés d'escriptura i lectura.
- *.rodata*: està compost per dades inicialitzades amb drets de lectura només.
- *.bss*: secció de dades sense inicialitzar amb accés a escriptura i lectura.

4.1.4.3 Tipus d'ELFs

Els fitxers ELF poden ser de dos tipus diferents en funció de com han estat vinculats: estàtics o dinàmics. L'etiquetatge fa referència al tipus de llibreries que estan essent utilitzades.

Els dinàmics són els que criden i afegeixen les llibreries que necessiten en temps d'execució, no estan incloses en el programa. Aquests, optimitzen l'execució al només carregar la llibreria necessària en el moment concís a canvi de reduir la portabilitat de l'execució.

En canvi, els estàtics ja porten inclosos totes les llibreries necessàries per a l'execució del fitxer, de forma que ofereixen molta més portabilitat que els dinàmics, però a canvi requereixen molt més espai de memòria.

4.1.5 Pyelftools i Capstone

Un cop generat el fitxer ELF, el següent pas es buscar i extreure la informació de cada segment necessària tant per carregar-la a la memòria del processador (.rodata, .bss), com per mostrarla en la interfície a l'usuari (.text). Aquesta tasca la podrem portar a terme amb l'ús de dues llibreries de python alhora: Pyelftools i Capstone.

Pyelftools [7] permet l'extracció i anàlisi del contingut dels fitxers ELF i DWARF. En el nostre cas que treballem amb un fitxer ELF el fem servir per iterar sobre els segments que conté l'ELF i extreure el contingut d'aquest, o de les seccions que conté. Després d'haver extret la informació que volem, si és necessari imprimir-la per pantalla necessitarem una eina que ens faciliti la conversió del llenguatge del ELF, bytecode, a un llenguatge que l'usuari pugui entendre, en aquest cas el llenguatge ensamblador (asm).

Capstone [6] és un framework que permet crear el codi ensamblador a partir d'un codi binari, i per tant desensemblar fitxers bytecode. És necessari especificar l'arquitectura (ARM, MIPS, RISC-V,...), el mode (32 o 64), i l'ordre dels bytes (little endian I big endian). Amb l'ús de Capstone podem convertir en llenguatge assemblador el contingut de les seccions que voldrem mostrar a l'usuari, i per tant aportar les funcionalitats de les qual Pyelftools manca.

En el nostre treball, s'han fet servir aquestes llibreries per dues funcionalitats diferents:

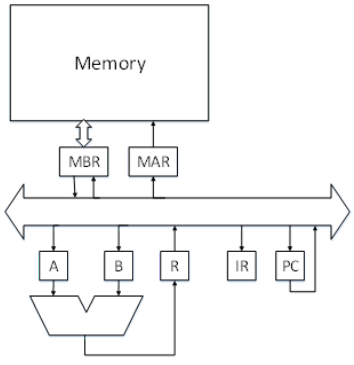
a) En el primer cas, per traduir el llenguatge d'alt nivell que l'usuari introdueix a llenguatge assemblador. Amb Pyelftools s'ha extret la secció .text i amb Capstone, definint les característiques del compilador (RISC-V 32 bits little endian), s'han traduït les tres dades més rellevants: adreça, instrucció, i instrucció en hexadecimal (il·lustració 8).

b) En el segon cas, s'ha iterat sobre tots els segments del fitxer ELF per extreure i carregar a la memòria del processador totes les seccions necessàries per a l'execució del programa.

4.2 Microprocessador RISC-V

En aquest treball s'ha implementat un microprocessador RISC-V microprogramat que està connectat a la interfície gràfica i on es podran executar codis i observar el seu funcionament. Tal i com s'ha esmentat en els objectius, aquest microprocessador serà sense pipeline i diferirà de l'estructura òptima per la qual està dissenyada la ISA RISC-V.

Aquest microprocessador estarà format per dos components: la unitat lògica-aritmètica (ALU) i la unitat de control (CU), i estan connectades entre elles i la memòria mitjançant un bus genèric de 32 bits. Estarà format per un únic bus genèric i per tant no hi haurà bus de dades, control i direccions per separat. Com es pot observar (il·lustració 1), aquest disseny no compta amb el registre d'estat un component que incorporen tots els microprocessadors moderns, i s'ha emulat la seva funcionalitat amb la memòria.



Il·lustració 1: arquitectura del microprocessador

4.2.1 Unitat de Control (CU)

La unitat de control [8] és l'encarregada de controlar i coordinar tots els components del microprocessador. Entre les seves funcions es troben: controlar la transferència de dades i instruccions incloses els dispositius d'E/S, i obtenir la instrucció a executar de memòria, interpretar-la i dirigir la seva execució. Entre els seus component dos dels més importants són el registre d'instruccions (IR) que s'encarrega de decodificar cada instrucció, i el comptador del programa (PC) que guarda

la posició en memòria de la següent instrucció.

4.2.1.1 Senyals de Control

Per portar a terme totes aquestes tasques, la CU té a la seva disposició diferents senyals de control connectades a la resta de components a través dels quals coordina l'execució. Per implementar aquests senyals de control, s'ha creat un diccionari on cada entrada està composta per una clau amb el nom del senyal i el seu valor, i per cada unitat i registre del microprocessador es defineixen les senyals de control necessàries per gestionar-lo. A tall d'exemple, en la il·lustració 2 es poden observar els senyals definides pel MBR.

- *mbr_e*: és el senyal de control que s'utilitza quan es vol comunicar al MBR que carregui les dades que rebrà, i per tant quan algun component envia dades amb destí el MBR aquest senyal es posa 1. El registre MBR cada vegada que es produeix un CLK comprova que aquest senyal de control estigui a 1, i si ho està actualitza el seu valor.

- *mbr_sel_mem* i *mbr_sel_r*: aquestes dos senyals identifiquen al MBR qui és l'emissor de les dades que ha de guardar.

```
CS['mbr_e'] = self.wire('mbr_e')
CS['mbr_sel_mem'] = self.wire('mbr_sel_mem')
CS['mbr_sel_r'] = self.wire('mbr_sel_r')
```

Il·lustració 2: Definició senyal de control MBR

Com el MBR només s'ha de preocupar de rebre dades de la memòria i el registre R, que s'encarrega d'emmagatzemar el resultat de la ALU, només necessita aquestes dos senyals de control. En altres registres com el MAR s'implementen més senyals de control, ja que pot rebre dades d'altres registres com l'R, el PC i els operands A i B.

4.2.1.2 Microcodi

El micro-codi [1] és una tècnica de disseny que interposa un capa d'organització informàtica entre el hardware de la CPU i el repertori d'instruccions del sistema que utilitza el programador. És una capa d'instruccions hardware que porten a terme les instruccions en llenguatge màquina de més alt nivell.

Per a la implementació d'aquest microprocessador s'ha creat un microcodi capaç d'executar les instruccions del ISA RISC-V més bàsiques implementades al ALU (4.2.2). En la il·lustració 3 es pot veure un tros d'aquest micro-codi un cop s'ha resetejat el comptador del programa (PC) a 0. El codi comença carregant al registre MAR la posició de la següent instrucció a memòria que conté el PC, es llegeix aquesta posició de memòria i es recupera el seu valor al MBR que s'envia al IR on es descodifica la instrucció a realitzar. S'espera un cicle de rellotge sense realitzar res i al següent CLK es comprova el tipus d'operació a realitzar, en funció de l'operació el PC adoptarà el valor de la posició del microcodi on es tracta aquesta. En aquest cas si resulta ser una operació de tipus addi (suma immediata) el comptador del programa passa a tenir el valor 8 i en el següent CLK s'executarà aquesta línia.

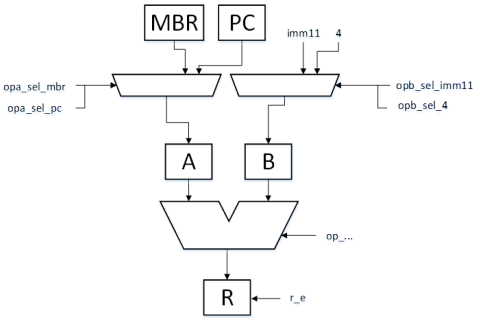
```
0: 'mar_e, mar_sel_pc',
1: 'mem_read',
2: 'mbr_e, mbr_sel_mem',
3: 'ir_e',
4: 'ins_decode',
5: 'nop',
6: 'EXECUTE',
7: 'goto 0',

# ADDI
8: 'mar_e, mar_sel_opa_idx',
9: 'mem_read',
10: 'mbr_e, mbr_sel_mem',
11: 'opa_e, opa_sel_mbr',
12: 'opb_e, opb_sel_se_imm12',
13: 'alu_add, r_e',
14: 'mar_e, mar_sel_opr_idx, mbr_e, mbr_sel_r',
15: 'mem_write',
16: 'goto 100',
```

Il·lustració 3: Tros de microcodi del microprocessador

4.2.2 Unitat aritmètica lògica (ALU)

La unitat aritmètica lògica o ALU [1] és el component del microprocessador encarregat de portar a terme les operacions de bits en les dades que rep de l'AC i el MBR, així com emmagatzemar-ne el resultat. Una ALU està composta, essencialment, per un sistema combinacional que combina diversos circuits lògics en un sol bloc, uns registres d'entrada, un registre acumulador, i un registre d'estats.

En aquest treball s'ha realitzat una implementació bàsica que està composta per dos registres d'entrada, un registre de sortida, i una emulació programada de les portes lògiques que componen els circuits lògics de l'ALU (il·lustració 4). Les portes lògiques en un processador són la combinació de transistors nmos i pmos formant transistors cmos. Una porta lògica és simplement un element que rep uns inputs, realitza algunes operacions, i produeix un output. Per exemple, una porta AND rep uns inputs, comprova bit a bit que els inputs estiguin a 1, i només si es compleix la condició posa el bit corresponent de l'output a 1.



Il·lustració 4: Esquema ALU bàsica

Per emular aquest comportament de forma programada s'han fet servir els operadors bitwise. Aquests operadors s'utilitzen com a qualsevol altre operador habitual, però la diferència radica en el fet que aquests no tracten les variables com a un sol valor únic, sinó com a un string de bits en complement 2. Per tant, si tornem a l'exemple de l'AND, per al cas de dos inputs, per emular el seu

funcionament es creen dues variables d'entrada que representaran el valor que tenen els inputs i un altre variable de sortida que representarà l'output. Després mitjançant l'operador bitwise '&' es realitza l'operació AND bit a bit entre les dues variables d'entrada i s'escriu el resultat a la de sortida.

Seguint el mateix esquema de creació de l'operador AND i mitjançant diferents operadors bitwise 2] s'han creat i emulat la resta d'operacions lògiques i aritmètiques que pot realitzar l'ALU.

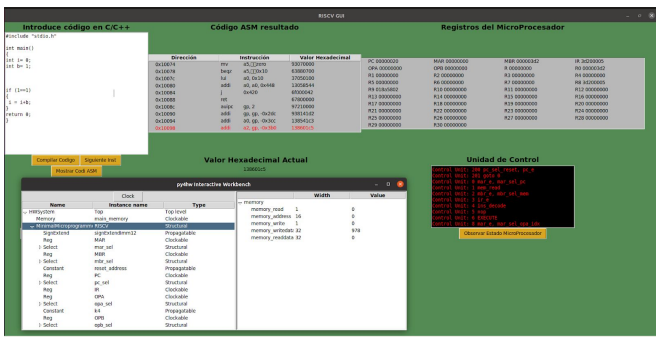
4.2.3 Memòria

La memòria és l'element on es guarda el programa a executar, les dades, i els resultats que s'obtinguin de les operacions a l'ALU. S'ha implementat seguint l'arquitectura Harvard [4] on les instruccions i les dades resideixen en la mateixa memòria. En l'inici d'aquest apartat 4.2 s'ha comentat que l'ALU d'aquest microprocessador microprogramat només compta amb registres d'entrada i sortida, i per tant no té el banc de registre d'estats habitual on s'emmagatzemen de forma temporal les dades que necessita l'ALU. Eventualment l'ALU necessita emmagatzemar dades que utilitzarà en futures instruccions, i per substituir la funcionalitat d'aquests registres s'han reservat les 32*4 primeres posicions de memòria que representen els 32 registres d'estats.

5 RESULTATS

5.1 INTERFÍCIE GRÀFICA

En la il·lustració 5 es pot apreciar el resultat de la interfície gràfic aconseguida. En els següents apartats mostrarem un a un els seus components amb una breu explicació del seu funcionament.

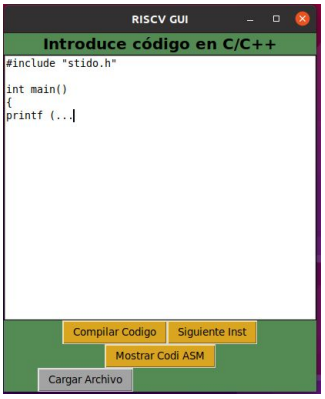


Il·lustració 5: Interfície gràfica

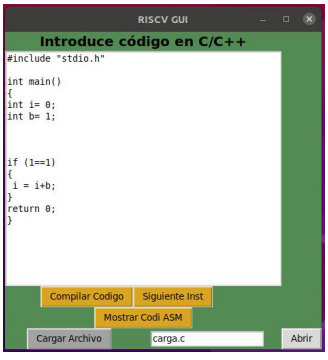
5.1.1 Quadre de Text

A l'executar la interfície gràfica el primer component que ens trobem és el quadre de text i els 4 botons que es poden apreciar en la il·lustració 6. En aquest quadre s'introdueix el codi en alt nivell que l'usuari vol executar en el microprocessador, i pot ser introduït de dos modes diferents: escrivint manualment el codi en el quadre, o carregant un fitxer.

Per carregar el fitxer, tenim un botó que destaca de color gris anomenat "Cargar Archivo" que al pressionar-lo s'obre un quadre de text per introduir el nom del fitxer (present en el mateix directori). Un cop introduït el nom del fitxer, apareix un altre botó amb nom "Abrir" que al ser pressionat carregarà el contingut del fitxer en el quadre de text (il·lustració 7).



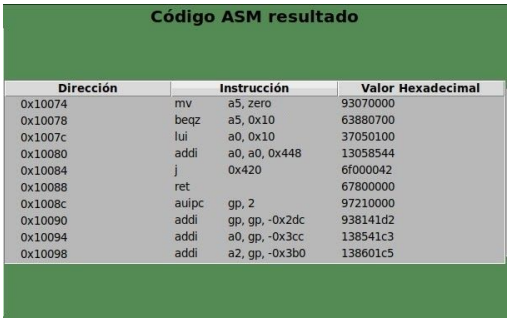
Il·lustració 6: Inserta manual del codi



Il·lustració 7: Càrrega del codi mitjançant un fitxer

5.1.2 Botons: Compilar i Mostrar codi

Un cop escrit o carregat el codi a executar, els següents passos es portaran a terme mitjançant els tres botons restants de color daurat. El primer botó "Compilar Codigo" serveix per compilar el codi introduït amb el compilador riscv32-unknown-elf-gcc que generarà el fitxer .elf. Després de generar el fitxer .elf, amb el botó "Mostrar Codi ASM" s'analitzarà el fitxer i, seguint el procediment esmentat a l'apartat 4.1.5, extreure'm de la secció ".text" el codi executable en llenguatge assemblador com es mostra a la il·lustració 8. El segon component per tant, és la impressió d'aquest codi assemblador.



Il·lustració 8: Codi executable en assemblador

5.1.3 Execució pas a pas del microprograma

El següent botó que podem trobar en la il·lustració 6 és "Siguiente Inst". La funcionalitat d'aquest botó està lligada al microprocessador i permet a l'usuari executar el microcodi pas a pas. Al prémer el botó s'enviarà un pols d'1 al rellotge (o clk) per avançar un cicle en el microprocessador. Al mateix temps, en el codi assemblador que es mostra en la interfície s'il·luminarà la instrucció asm que s'està executant actualment (il·lustració 9).

Il·lustració 9: Execució instrucció en instrucció

5.1.4 Registres microprocessador

El tercer component d'aquesta interfície és una capsa que conté els registres del microprocessador i que s'actualitzen a temps real als valors que contenen en cada cicle de rellotge (il·lustració 10). El procés per actualitzar-los que se segueix és accedir als seus valors en el microprocessador (PC, MAR, etc) i a la memòria (RX), i recuperar-los cada vegada que es posi a 1 el clk.

Il·lustració 10: Registres del microprocessador

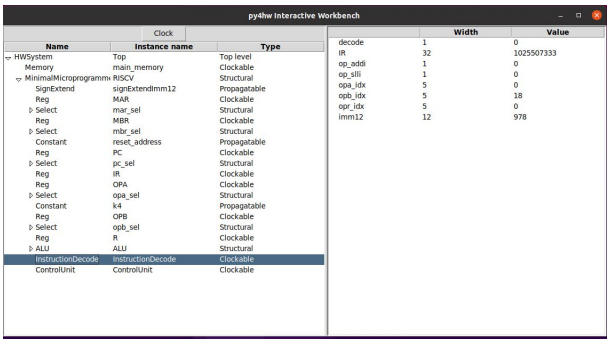
5.1.5 Microcodi

Aquest component s'encarrega de mostrar les microinstruccions que va executant la unitat de control. A mesura que avancen els cicles de rellotge es va imprimint la microinstrucció actual (il·lustració 11).

Il·lustració 11: Microcodi de la UC

5.1.6 Estat del microprocessador

El següent i últim component de la interfície és un botó amb el nom "Observar Estado MicroProcesador" que es pot apreciar en la il·lustració 1 a sota dels registres. Aquest botó està vinculat al microcontrolador i al ser premut desplega una nova interfície gràfica mostrant totes les dades i valors que pren la memòria, els registres, l'ALU, la unitat de control, i la resta components del microprocessador (il·lustració 12).



Il·lustració 12: Interfície gràfica del microprocessador

6 CONCLUSIÓNS

Un cop finalitzat el següent treball, es considera que s'han pogut assolir els objectius més elementals per fer d'aquesta una eina complementaria a l'estudi.

S'ha pogut implementar un microprocessador RV32I microprogramat amb una ALU capaç de fer les operacions més bàsiques. S'han creat les diferents connexions amb la resta de components per permetre una coordinació i gestió correcta per part de la unitat de control, s'ha emulat el funcionament de les portes lògiques de l'ALU amb èxit per portar a terme les operacions simples de suma, resta i comparació, i també s'ha implementat una memòria principal on resideix el codi a executar que introdueix l'usuari a través de la interfície gràfica. En aquesta memòria, també s'ha implementat la funcionalitat dels registres d'estat, que normalment es troben al processador, que auxilien a l'ALU per emmagatzemar dades de forma temporal.

D'altra banda, s'ha pogut crear una interfície gràfica que permet explorar de forma interactiva el procés que inclou des de l'escriptura de codi d'alt nivell fins a l'execució del

microprograma pas a pas en el microprocessador. S'han implementat diverses finestres que permeten a l'usuari observar el contingut que tenen els diversos components del microprocessador: la finestra dels registres on l'usuari pot seguir el recorregut que realitza el flux de dades i instruccions durant l'execució, i quins registres s'utilitzen per a cada microinstrucció. Un altre és la de la unitat de control (CU) que mostra quina microinstrucció del microprograma està executant actualment la CU, i fent ús de la finestra de registre l'usuari pot identificar com reacciona cada registre a aquestes. També, es pot trobar una finestra amb el codi d'alt nivell traduït a llenguatge asm, juntament amb la seva posició a la memòria del microprocessador i el seu valor hexadecimal. En aquesta podem identificar el valor hexadecimal de les instruccions que s'executen en el microprocessador, i per tant seguir aquest valor des que es recuperà de memòria tot el recorregut que segueix mitjançant la finestra de la CU i dels registres fins que es tornà a emmagatzemar. I finalment, trobem una última que s'obre mitjançant el botó d'observar estat del microprocessador, on es pot observar tant la memòria com el microprocessador i identificar cada component amb el seu nom, el seu tipus, la seva mida i el seu valor actual.

A causa de compilacions en les diferents implementacions d'aquest treball, no s'ha pogut acabar de complir l'objectiu de poder executar pas per pas les instruccions asm del codi. Com a línia futura seria interessant ampliar la funcionalitat d'aquesta interfície per ser capaç d'executar d'una en una les instruccions asm. Un altre, seria la implementació d'un esquema del microprocessador en la interfície amb tots els senyals que comuniquen els components, i per cada microinstrucció que s'executi s'il·luminin els senyals implicades.

AGRAÏMENTS

M'agradaria agrair a la meva família per tot el suport i ajuda que m'han donat al llarg de tota aquesta carrera. A l'Andra per la constant ajuda en tots els dubtes que m'han sorgit durant la redacció d'aquest treball i pel seu suport anímic. En Soriano per ensenyar-me el veritable significat de ser un *ball chaser* i per compartir una mica de les seves grans habilitats d'edició.

També, m'agradaria agrair al David per tot el compromís que ha tingut en aquest treball. Per l'organització de les reunions setmanals que han sigut clau en el desenvolupament. Sempre ha estat atent als correus que li enviava i ben disposat a respondre tots els dubtes que m'han sorgit.

BIBLIOGRAFIA

[1] Barriga, A. (s. d.). RISC-V processors design: a methodology for cores development. *Instituto de Microelectrónica de Sevilla.*

[2] BitwiseOperators. (2013). FAQ: What do the operators <<, >>, &, |, ~, and ^ do? *Python*. https://wiki.python.org/moin/BitwiseOperators

[3] Costa, C. D. (2020). Top 10 Python GUI Frameworks for Developers. *Towards Data Science*. https://towardsdatascience.com/top-10-python-gui-frameworks-for-developers-adca32fbe6fc

[4] Darshan. Institute of Engineering & Technology. (s. d.). Unit 1. Introduction to microprocessor. http://www.darshan.ac.in/Upload/DIET/Documents/CE/2150707-MPI-Study-Material_04112017_033410AM.pdf

[5] GeeksforGeeks. (2019). Introduction of control Unit and its Design. https://www.geeksforgeeks.org/introduction-of-control-unit-and-its-design/

[6] Github. (2021). Aquynh/Capstone. https://github.com/aquynh/capstone/tree/next

[7] Github. (2021). Eliben/pyelftools. https://github.com/eliben/pyelftools

[8] Lecture 2. The CPU, Instruction Fetch & Execute. https://www.robots.ox.ac.uk/~dwm/Courses/2CO_2014/2CO-N2.pdf

[9] Powell, A. (1997). Web 101: A History of the GUI. *Wired*. https://www.wired.com/1997/12/web-101-a-history-of-the-gui/#:~:text=In%201979%2C%20the%20Xerox%20Palo,first%20prototype%20for%20a%20GUI.&text=When%20Jobs%20saw%20this%20prototype,expensive%3B%20no%20one%20bought%20it

[10] PyQT. (2020). About PYQT. *Python*. https://wiki.python.org/moin/PyQt

[11] Python Software Foundation. (2021). Tkinter - Python interface to Tcl/Tk. *Python*. https://docs.python.org/3/library/tkinter.html

[12] RISC-V. (2020). https://riscv.org/

[13] Tool Interface Standards. (s.d.). Object Files. *Executable and Linkable Format (ELF)*. (pp. 1-21). http://www.skyfree.org/linux/references/ELF_Format.pdf

[14] Wikipedia. (2020). Kivy (framework). https://en.wikipedia.org/wiki/Kivy_(framework)