پروژه درس هم‌طراحی سخت‌افزار و نرم‌افزار ترم 14002 مهلت تحویل: 10 تیر 1401

نام و نام خانوادگی : سبا عامری شماره دانشجویی: 970122680015

نام و نام خانوادگی : معصومه مختاری شماره دانشجویی: 970122681006

در این پروژه، پردازنده 6 بیتی که در کلاس طراحی شده را پیاده‌سازی کرده و برای آن برنامه‌نویسی می‌کنیم.

توجه: این پروژه در صورتی قابل قبول است که برای آن گزارش هم نوشته شود. در این گزارش نحوه پیاده‌سازی پردازنده و اجرای برنامه توسط آن با استفاده از عکس‌های مناسب از خروجی شبیه‌سازی نشان داده شود.

بخش اول (٪40 نمره پروژه): برای انجام این پروژه ابتدا پردازنده را با استفاده از VHDL یا Verilog پیاده‌سازی کرده و صحت عملکرد آن را با اجرای کد زیر که دو عدد 7 و 4 را با هم جمع می‌کند بررسی کنید.

```
LOAD R0, 7
LOAD R1, 4
ADD R0, R1
```

بخش دوم (٪20 نمره پروژه): با توجه به این‌که این پردازنده دستور ضرب ندارد، عمل ضرب را با استفاده از عمل جمع و به صورت نرم‌افزاری پیاده‌سازی کرده و صحت عملکرد آن را با یک مثال نشان دهید (مشابه بخش اول یک کد اسمبلی بنویسید که عمل ضرب را انجام دهد). به عنوان مثال، حاصلضرب عدد 8 در 6 را حساب کند.

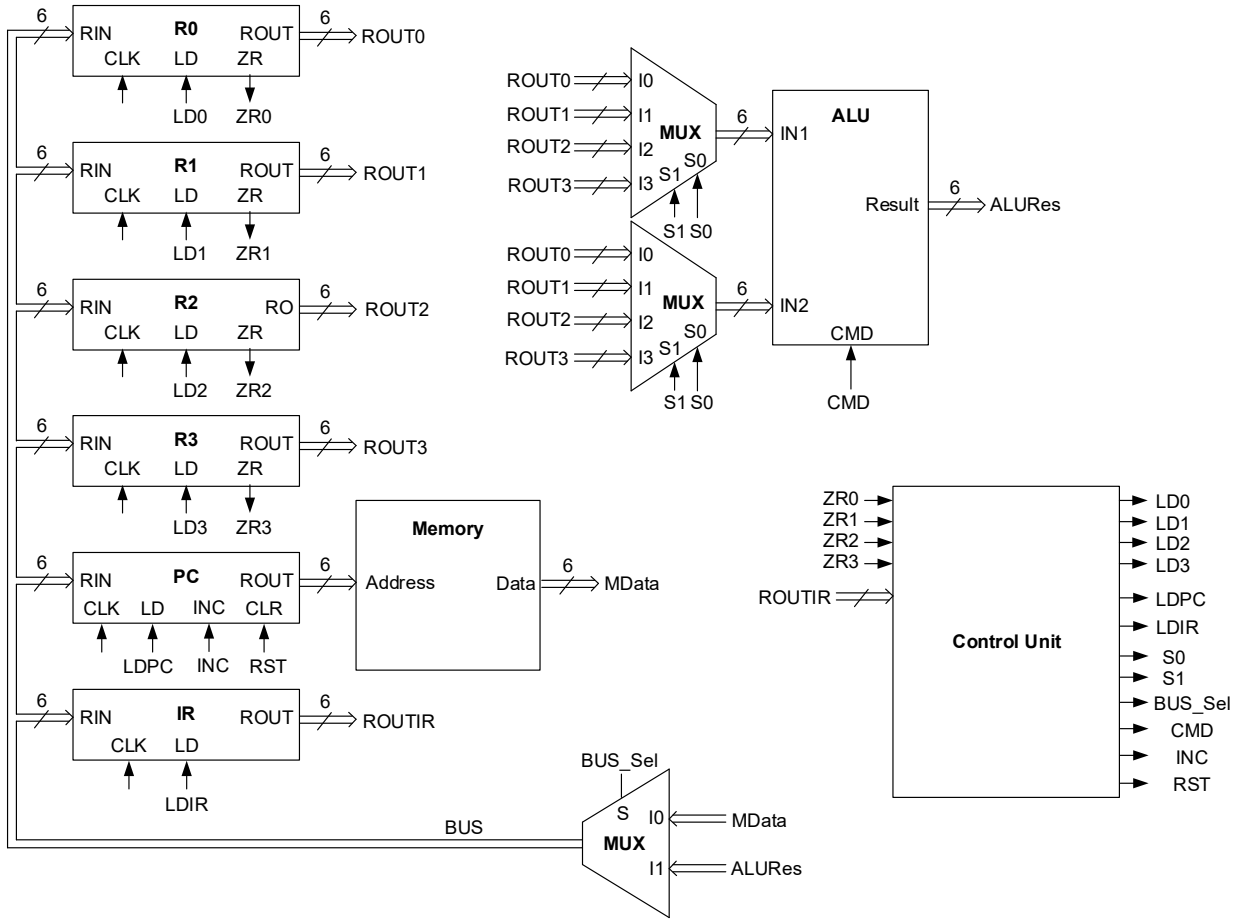
بخش سوم (٪40 نمره پروژه): دستور ضرب را با کمترین سربار سخت‌افزاری به مجموعه دستورات اضافه کرده و صحت عملکرد آن را با نوشتن یک کد که حاصلضرب 8 در 6 را حساب کند نشان دهید. توجه کنید که برای این کار نیاز است تغییراتی در سخت‌افزار و کد دستورات ایجاد کنید.

معماری پردازنده:

پروژه درس هم‌طراحی سخت‌افزار و نرم‌افزار ترم 14002 مهلت تحویل: 10 تیر 1401

نام و نام خانوادگی : سبا عامری شماره دانشجویی: 970122680015

نام و نام خانوادگی : معصومه مختاری شماره دانشجویی: 970122681006



دستورات پردازنده:

این پردازنده چهار دستور LOAD، ADD، SUB و JNZ با کد دستور (Op Code) زیر است:

دستور	کد دستور
LOAD	00
ADD	01
SUB	10
JNZ	11

قالب دستورات:

Op Code	R_{SRC}	R_{DST}

پروژه درس هم‌طراحی سخت‌افزار و نرم‌افزار ترم 14002 مهلت تحویل: 10 تیر 1401

نام و نام خانوادگی : سبا عامری شماره دانشجویی: 970122680015

نام و نام خانوادگی : معصومه مختاری شماره دانشجویی: 970122681006

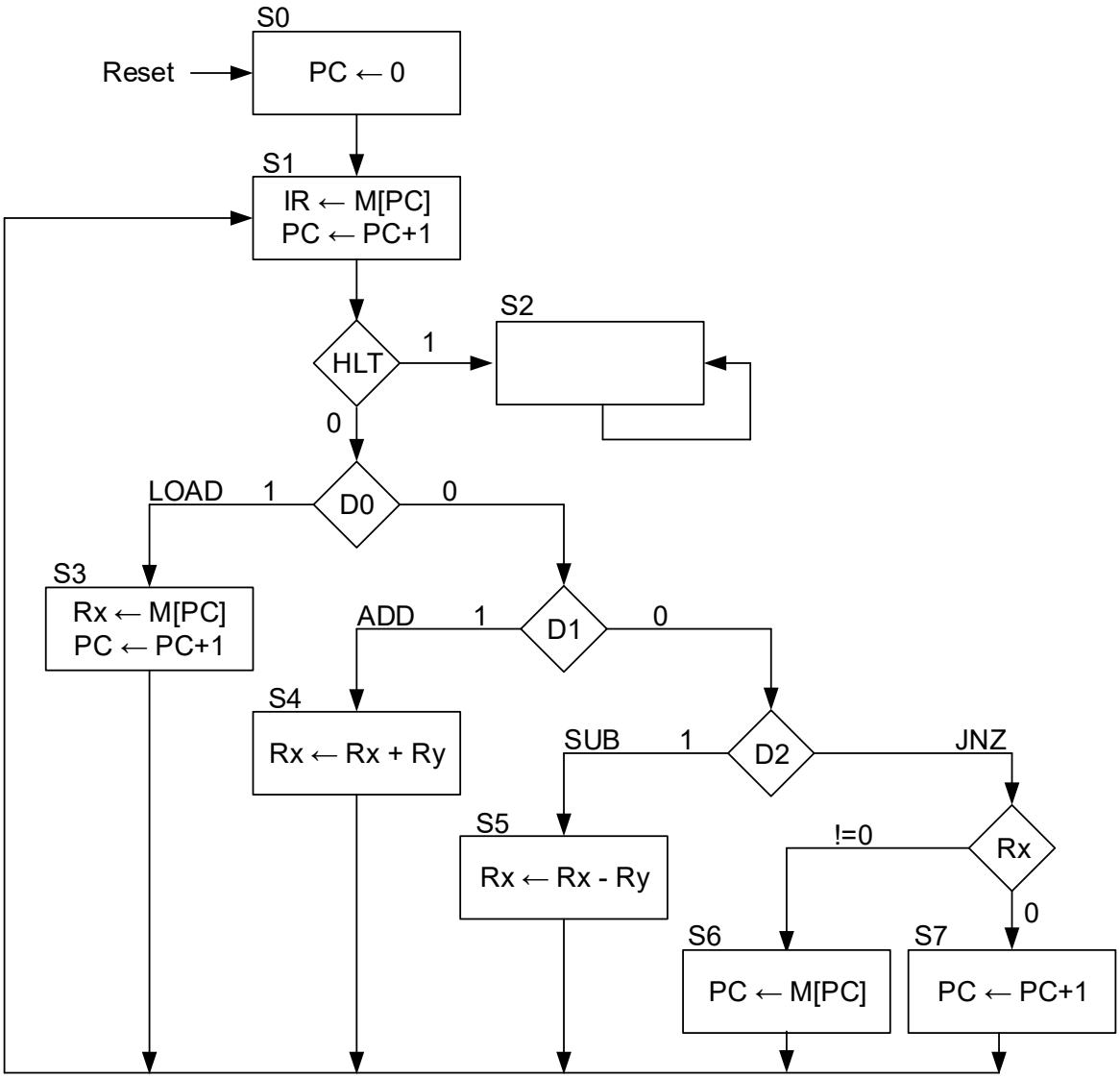
اسمبلی دستور	RTL	چینش در حافظه
LOAD Rx, VALUE	Rx ← M[PC]	PC → 00 Rx 00 / مقدار / دستور بعدی
ADD Rx, Ry	Rx ← Rx + Ry	PC → 01 Rx Ry / دستور بعدی
SUB Rx, Ry	Rx ← Rx - Ry	PC → 10 Rx Ry / دستور بعدی
JNZ Rx, Address	If (Rx != 0) PC ← M[PC] else PC ← PC + 1	PC → 11 Rx 00 / آدرس پرش / دستور بعدی

پروژه درس هم‌طراحی سخت‌افزار و نرم‌افزار ترم 14002 مهلت تحویل: 10 تیر 1401

نام و نام خانوادگی : سبا عامری شماره دانشجویی: 970122680015

نام و نام خانوادگی : معصومه مختاری شماره دانشجویی: 970122681006

چارت ASM برای طراحی واحد کنترل:



پروژه درس هم‌طراحی سخت‌افزار و نرم‌افزار ترم 14002 مهلت تحویل: 10 تیر 1401

نام و نام خانوادگی : سبا عامری شماره دانشجویی: 970122680015

نام و نام خانوادگی : معصومه مختاری شماره دانشجویی: 970122681006

بخش اول:

در قسمت نرم افزار کد اسمبلی نوشته شده را از فایل تکست خوانده، به یک لیست تبدیل میکنیم برای مثال ورودی ما:

```
LOAD R0, 0
LOAD R1, 10
LOAD R2, 2
L0: ADD R0, R1
SUB R1, R2
JNZ R1, L0
HLT
```

به لیست زیر تبدیل می‌شود:

```
['LOAD', 'R0', '0'], ['LOAD', 'R1', '10'], ['LOAD', 'R2', '2'], ['ADD', 'R0', 'R1'],
['SUB', 'R1', 'R2'], ['JNZ', 'R1', 'L0'], ['HLT']
```

یک تابع برای تشخیص و ذخیره لیبل در لیست دیگر اجرا می‌کنیم لیبل و موقعیتش را در این تابع ذخیره می‌کنیم به ازای تابع های دو خطی مثل Load، 2 عدد و به ازای تابع های یک خطی 1 عدد

```
[['L0', 6]]
```

سپس لیست را به کدهای باینری تبدیل کرده و در فایل متن ذخیره می‌کنیم

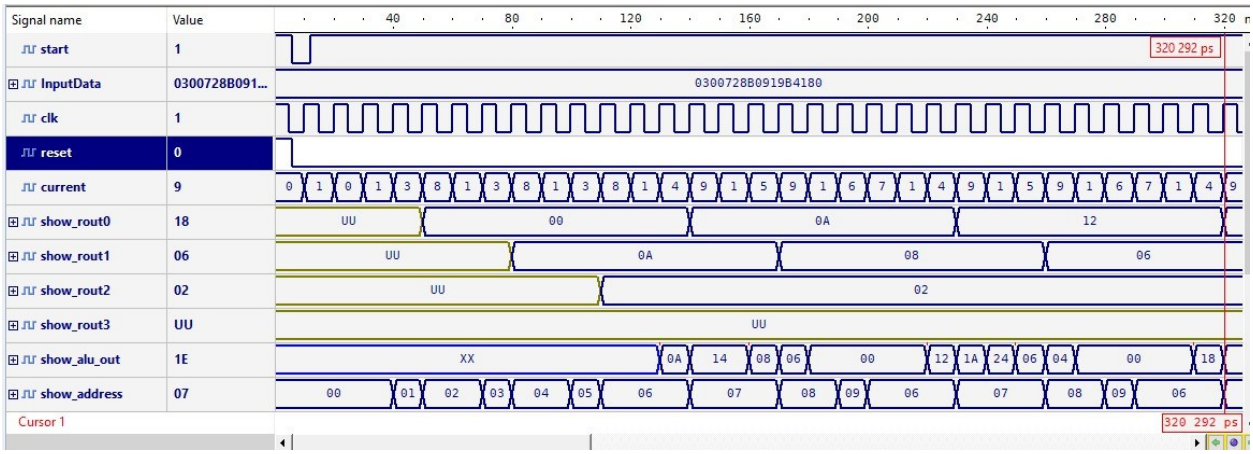
```
['000011', '000000', '000111', '001010', '001011', '000010', '010001',
'100110', '110100', '000110', '000000']
```

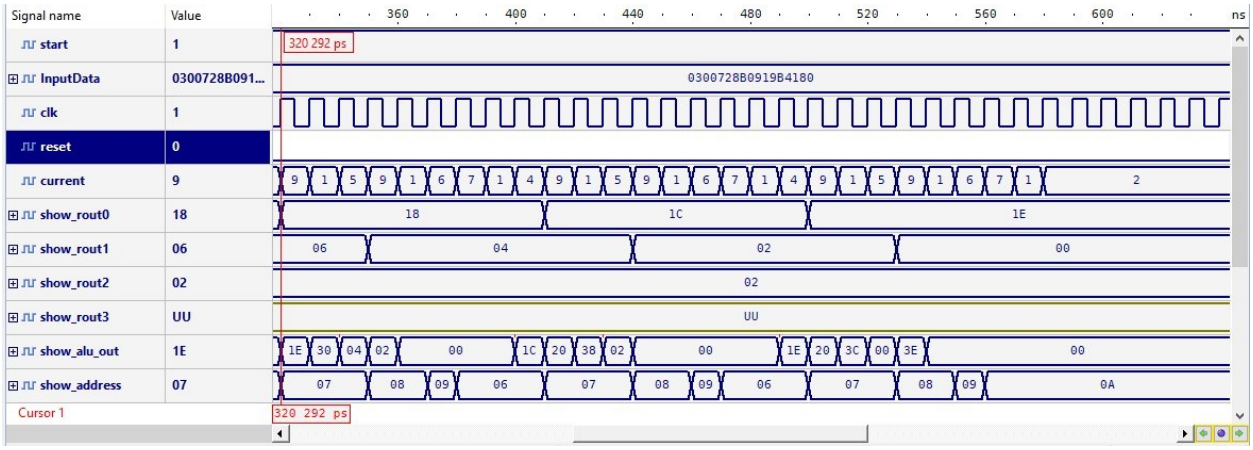
در بخش اول پروژه همه قسمت های مختلف پردازنده را بصورت جدا پیاده سازی کرده، یک فایل پروسسور ایجاد کرده و و از هر قسمت نمونه گیری انجام می‌دهیم. با استفاده از سیگنال ها قسمت های مختلف پردازنده را به یکدیگر متصل می‌کنیم. در مرحله اول کد باینری را در مموری قرار می‌دهیم و پروسسور را ریست می‌کنیم. کنترل یونیت دستور شروع شدن pc را می دهد. و دستور های مختلف خوانده شده و در کنترل یونیت استیت های مختلفی داریم که هر استیت مربوط به دستور خاصی است. در این بخش توانایی انجام دستور های load, add, sub, jump را داریم . برای مثال خروجی دستور بالا بصورت زیر خواهد بود.

پروژه درس هم‌طراحی سخت‌افزار و نرم‌افزار ترم 14002 مهلت تحویل: 10 تیر 1401

نام و نام خانوادگی : سبا عامری شماره دانشجویی: 970122680015

نام و نام خانوادگی : معصومه مختاری شماره دانشجویی: 970122681006





بعد از اتمام دستور ها در رجیستر شماره صفر مقدار 30 ، در ریجستر یک مقدار صفر و در رجیستر دو مقدار 2 ذخیره خواهد شد.

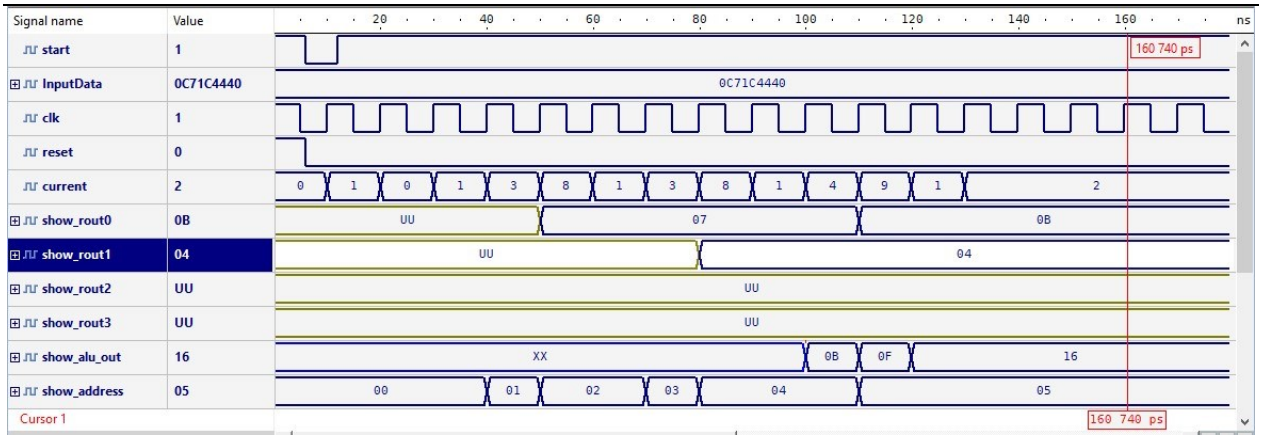
خروجی دستور زیر نیز به این صورت خواهد بود.

```
LOAD R0, 7
LOAD R1, 4
ADD R0, R1
```

پروژه درس هم‌طراحی سخت‌افزار و نرم‌افزار ترم 14002 مهلت تحویل: 10 تیر 1401

نام و نام خانوادگی : سبا عامری شماره دانشجویی: 970122680015

نام و نام خانوادگی : معصومه مختاری شماره دانشجویی: 970122681006



بخش دوم:

در بخش دوم دستور ضرب را بصورت نرم افزاری اجرا می‌کنیم

```
load r1, 6
load r2, 8
smul r1, r2
hlt
```

قبل از اینکه دستور ها را مانند بخش قبلی به کد باینری تبدیل کنیم چک می‌کنیم که آیا دستور ضرب نرم افزاری را می‌توانیم بیابیم در صورت دیدن دستور `smul` آن را به چندین دستور جمع ،تفریق و پرش اسمبلی تبدیل کرده سپس این دستور ها را مانند بخش قبلی به کد باینری تبدیل می‌کنیم.

هر دستور ضرب یک حلقه ایجاد می‌کند که در این حلقه هر دور عدد دوم را منهای 1 کرده تا موقعی که صفر شود. و درون حلقه عدد اول را با مقدار اولیه خودش جمع کرده، در رجیستر عدد اول ذخیره می‌کنیم. دو رجیستر برای عدد اول و دوم استفاده می‌شود. در دو رجیستر خالی دیگر عدد 1 و عدد اول دستور ضرب را ذخیره کرده تا هر بار با عدد اول دستور ضرب جمع کنیم.

```
[['load', 'r1', '6'], ['load', 'r2', '8'], ['LOAD', 'R0', '1'], ['LOAD',
'R3', '0'], ['ADD', 'R3', 'R1'], ['SUB', 'R2', 'R0'], ['ADD', 'R1', 'R3'],
['SUB', 'R2', 'R0'], ['JNZ', 'R2', 'LL2'], ['hlt']]


[['LL2', 10]]
```

پروژه درس هم‌طراحی سخت‌افزار و نرم‌افزار ترم 14002 مهلت تحویل: 10 تیر 1401

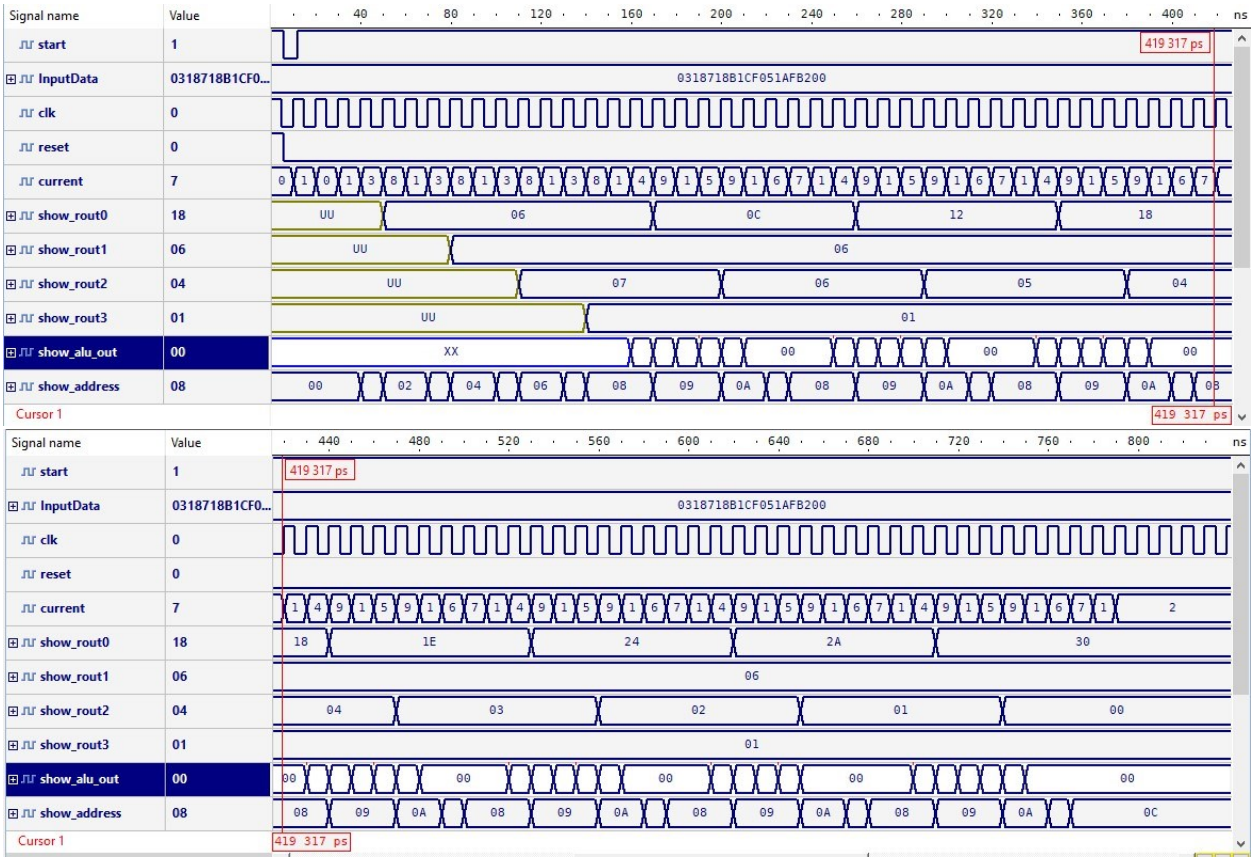
نام و نام خانوادگی : سبا عامری شماره دانشجویی: 970122680015

نام و نام خانوادگی : معصومه مختاری شماره دانشجویی: 970122681006

کد باینری خروجی این برنامه

```
['000111', '000110', '001011', '001000', '000011', '000001', '001111',
'000000', '011101', '101000', '010111', '101000', '111000', '001010',
'000000']
```

سپس این کد باینری را بصورت ورودی به سخت افزار می‌دهیم



بخش سوم:

در این بخش قصد داریم با کمترین سربار سخت افزاری دستور ضرب را ایجاد کنیم.

در این بخش برای خواندن دستور ضرب (100) مجبور هستیم تا دستور ها را 7 بیتی بکنیم.

در قسمت نرم افزاری دستورها همه 7 بیتی شده و دستور ضرب را نیز همانند دستور جمع و تفریق به کد باینری تبدیل می‌کنیم.

پروژه درس هم‌طراحی سخت‌افزار و نرم‌افزار ترم 14002 مهلت تحویل: 10 تیر 1401

نام و نام خانوادگی : سبا عامری شماره دانشجویی: 970122680015

نام و نام خانوادگی : معصومه مختاری شماره دانشجویی: 970122681006

```
load r0, 6
load r1, 8
mul r1, r0
hlt
```

لیست:

```
[['load', 'r0', '6'], ['load', 'r1', '8'], ['mul', 'r1', 'r0'], ['hlt']]
```

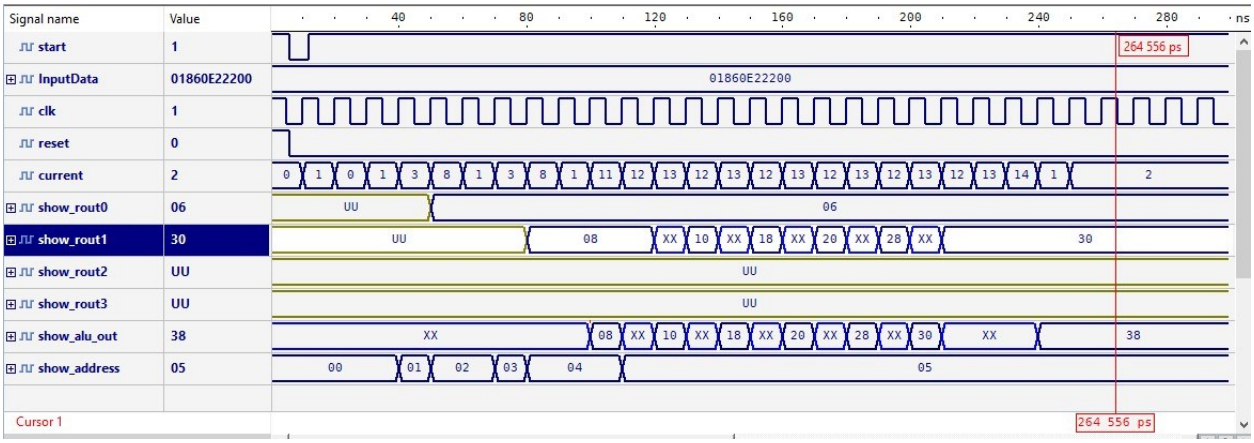
لیست کد باینری:

```
['0000011', '0000110', '0000111', '0001000', '1000100', '0000000']
```

از آنجایی که در دستور ضرب از دستور جمع استفاده می‌شود، برای اینکه کمترین سربار سخت افزاری را داشته باشیم می‌خواهیم از همان دستور جمع موجود در `alu` استفاده کنیم. به همین دلیل `alu` موجود در بخش های قبلی را بصورت `explicit` انجام می‌دهیم. سپس برای اجرای دستور ضرب به تعداد عدد دوم این دستور عدد اول را جمع می‌کنیم.

از آنجایی که مقدار جمع در رجیستر عدد اول دستور ذخیره شده و مقدارش تغییر می‌کند، به یک رجیستر نیازمندیم تا مقدار اولیه عدد اول دستور را در خود ذخیره کند. سپس با استفاده از سیگنال ها و ایجاد استیت های جدید می‌توانیم دستور ضرب را انجام بدهیم.

این دستور در `alu` دارای دو مرحله خواهد بود در مرحله اول (`cmd="11"`) خروجی `alu` را عدد اول قرار می‌دهیم. بصورت همزمان در رجیستر دستور ضرب نیز این مقدار ذخیره می‌شود. در مرحله دوم (`cmd="10"`) عدد اول دستور با مقدار درون رجیستر اول جمع شده و در رجیستر اول ذخیره می‌شود. این مرحله به اندازه عدد دوم دستور ضرب تکرار شده سپس به کنترل یونیت سیگنال اتمام دستور ضرب ارسال می شود تا کنترل یونیت به دستور بعدی برود.



پروژه درس هم‌طراحی سخت‌افزار و نرم‌افزار ترم 14002 مهلت تحویل: 10 تیر 1401

نام و نام خانوادگی : سبا عامری شماره دانشجویی: 970122680015

نام و نام خانوادگی : معصومه مختاری شماره دانشجویی: 970122681006

جدول استیت های درون کنترل یونیت:

شماره استیت	دستور مربوطه	توضیحات
0	Reset	وقتی ریست اتفاق می افتد به این استیت می‌رویم
1	Choose state	دستور خوانده شده و به استیت مورد نظر خواهیم رفت
2	Halt	کنترل یونیت همینجا بافی خواهد ماند و دستور دیگر را اجرا نمی‌کند
3	Load: part 1	لود رجیستر درون دستور 1 خواهد شد
4	Add: part 1	لود رجیستر اول دستور 1 خواهد شد تا جواب در آن ذخیره شود مقدار cmd دستور جمع به آن داده می‌شود
5	Sub: part 1	لود رجیستر اول دستور 1 خواهد شد تا جواب در آن ذخیره شود مقدار cmd دستور تفریق به آن داده می‌شود
6	Jump: part 1	چک می‌کنیم آیا رجیستر درون دستور مقدار صفر را دارد یا خیر در صورت اینکه صفر باشد لود pc را 1 می‌کنیم
7	Jump: part 2	دستور پرش دارای دو خط است اگر قرار بود پرش انجام بگیرد وقتی در این استیت هستیم به Pc مقدار مورد نظر داده می‌شود. در این استیت لود pc را صفر می‌کنیم
8	Load: part 2	دستور لود دارای دو خط است. در این استیت در رجیستر دستور مقدار مورد نظر لود خواهد شد. مقدار لود همه رجیسترها را صفر می‌کنیم.
9	Add/Sub: part 2	نتیجه جمع را تفریق بدست می‌آید. از آنجایی که alu را بصورت explicit انجام داده ایم برای اینکه جواب مشخص شود به دو کلاک نیازمند است. این استیت در بخش های قبلی استفاده نمی‌شود
10	Add/Sub: part 3	جواب در رجیستر اول دستور ذخیره شده سپس لود همه رجیستر ها را صفر می‌کنیم
11	Multiply: part 1	رجیستر هایی که ضرب می‌شوند را لود کرده رجیستری که مقدار اولیه را ذخیره میکند را نیز لود میکنیم و ورودی alu را مشخص میکنیم
12	Multiply: part 2	عمل جمع رجیستر اول با مقدار ذخیره شده را انجام میدهیم
13	Multiply: part 3	در این استیت چک می‌کنیم که آیا alu سیگنال اتمام دستور ضرب را ارسال کرده است یا خیر تا موقعی که سیگنال اتمام ضرب 1 نشود به استیت 12 برمیگردیم وقتی 1 شد به استیت 14 خواهیم رفت
14	Multiply: part 4	در این استیت جواب ضرب ذخیره می‌شود