پردازنده‌ای با مشخصات زیر در نظر بگیرید:

- رجیستر فایل شامل ۸ رجیستر ۸ بیتی است که آنها را *r0* تا *r7* می‌نامیم. رجیستر *r0* همواره مقدار ثابت ***صفر*** را در خود ذخیره کرده است.
- تمام دستورات این پردازنده ۱۹ بیتی هستند.
- پردازنده دارای یک حافظه دستور با گنجایش ۴۰۹۶ دستور (۱۹ بیتی) و یک حافظه داده با گنجایش ۲۵۶ بایت است.
- پردازنده دارای یک استک با گنجایش ۸ خانه برای ذخیره کردن آدرس برگشت فراخوانی تابع است.
- پردازنده دارای دو فلیپ‌فلاپ C و Z است.
- پردازنده دارای یک ALU برای انجام محاسبات ۸ بیتی است. ALU دارای دو ورودی ۸ بیتی A و B، یک ورودی یک بیتی Cin، یک خروجی ۸ بیتی Y، یک خروجی یک بیتی Zero و یک خروجی یک بیتی Cout است. خروجی Zero در صورتی که مقدار Y=0 باشد برابر ۱ و در غیر این‌صورت برابر ۰ است. ورودی و خروجی Cin و Cout به ترتیب بیت نقلی ورودی و بیت نقلی خروجی عملیات محاسباتی است.

قالب دستورات این پردازنده در زیر آمده است:

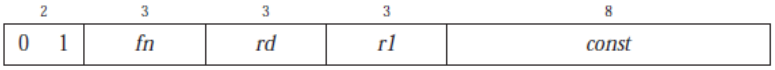
۱- دستورات محاسباتی/منطقی رجیستری

2		3	3	3	3	5				
0	0	*fn*	*rd*	*r1*	*r2*	*x*	*x*	*x*	*x*	*x*

یک عملیات محاسباتی/ منطقی روی دو رجیستر (*r1, r2*) انجام شده و نتیجه در رجیستر مقصر (*rd*) ذخیره می‌شود. نوع عملیات توسط *fn* مشخص می‌شود (جدول زیر را ببینید). اگر نتیجه محاسبه برابر ***صفر*** باشد *Z=1* و در غیر این‌صورت *Z=0*. برای دستورات *ADD* و *ADDC* مقدار *C* برابر بیت نقلی عملیات جمع و برای دستورات *SUB* و *SUBC* مقدار *C* برابر بیت قرضی عملیات تفریق است. برای دستورات منطقی همواره *C=0*.

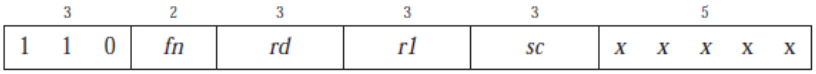
Instruction	Operation	fn
ADD	Add without carry in	000
ADDC	Add with carry in	001
SUB	Subtract without carry in	010
SUBC	Subtract with carry in	011
AND	Bitwise logical and	100
OR	Bitwise logical or	101
XOR	Bitwise logical xor	110
MASK	Bitwise logical mask (and-not)	111

۲- دستورات محاسباتی/منطقی بلافصل (Immediate)



یک عملیات محاسباتی/ منطقی روی یک رجیستر (*r1*) و یک داده بلافصل (*const*) انجام شده و نتیجه در رجیستر مقصر (*rd*) ذخیره می‌شود. نوع عملیات و نحوه تغییر *C* و *Z* مانند دستورات محاسباتی/منطقی رجیستری است.

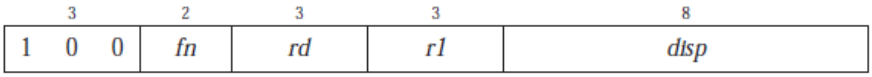
۳- دستورات شیفت



یک عملیات شیفت و یا دوران روی یک رجیستر (*r1*) به تعداد بیت‌های مشخص شده (*sc*) انجام شده و نتیجه در رجیستر مقصر (*rd*) ذخیره می‌شود. نوع عملیات توسط *fn* مشخص می‌شود (جدول زیر را ببینید). اگر نتیجه محاسبه برابر ***صفر*** باشد *Z=1* و در غیر این‌صورت *Z=0*. مقدار C برابر بیتی می‌شود که از عملیات شیفت یا دوران بیرون ریخته می‌شود.

Instruction	Operation	fn
SHL	Shift left	00
SHR	Shift right	01
ROL	Rotate left	10
ROR	Rotate right	11

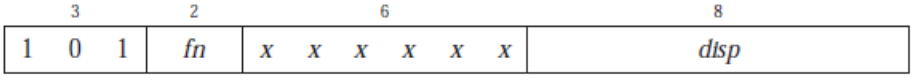
۴- دستورات حافظه



برای عملیات خواندن(نوشتن) از (در) یک محل حافظه به کار می‌رود. آدرس محل حافظه از جمع یک مقدار علامت‌دار هشت بیتی (*disp*) و یک رجیستر پایه (*r1*) به دست می‌آید. برای دستور *LDM* رجیستر *rd* رجیستر مقصد و برای دستور *STM* رجیستر *rd* یکی از اپرندهای دستور است. نوع عملیات توسط *fn* مشخص می‌شود (جدول زیر را ببینید). مقادیر *C* و *Z* با اجرای این دستورات تغییر نمی‌کنند.

Instruction	Operation	fn
LDM	Load memory	00
STM	Store memory	01

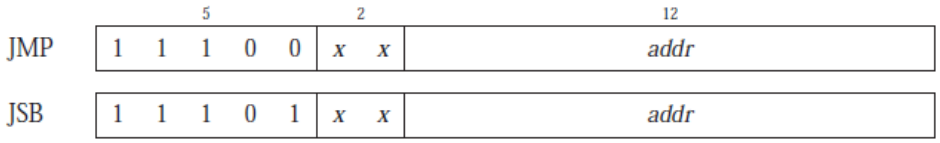
۵- دستورات پرش شرطی



در صورت برقرار بودن شرط پرش، پرش به آدرس پرش انجام می‌شود و در غیر این‌صورت *pc* با ۱ جمع می‌شود تا به سراغ دستور بعد از دستور پرش برویم. آدرس پرش از جمع یک مقدار علامت‌دار هشت بیتی (*disp*) و آدرس دستور بعد از پرش (*pc+1*) به دست می‌آید. نوع عملیات توسط *fn* مشخص می‌شود (جدول زیر را ببینید). مقادیر *C* و *Z* با اجرای این دستورات تغییر نمی‌کنند.

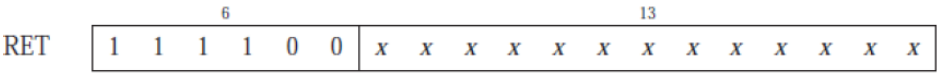
Instruction	Operation	fn
BZ	Branch if zero	00
BNZ	Branch if not zero	01
BC	Branch if carry	10
BNC	Branch if not carry	11

۶- دستورات پرش بدون شرط



با اجرای این دستورات، کنترل به آدرس مشخص شده (*addr*) منتقل می‌شود. دستور *JSB* برای فراخوانی تابع به کار می‌رود. در این دستور پیش از این که پرش انجام شود، ابتدا آدرس برگشت (*pc+1*) روی استک *push* می‌شود. مقادیر *C* و *Z* با اجرای این دستورات تغییر نمی‌کنند.

۷- دستورات متفرقه



دستور *RET* برای بازگشت از یک تابع به کار می‌رود. برای این منظور آدرس بازگشت از روی استک *pop* شده و در *pc* قرار می‌گیرد.

این پردازنده دارای یک پایه ورودی *reset* است. در صورت فعال شدن این پایه، مقادیر فلیپ‌فلاپ‌های Z و C صفر شده و سپس با صفر شدن *pc* اجرای دستورات از آدرس *000* حافظه شروع می‌شود.

مسیر داده و واحد کنترل این پردازنده را به صورت پایپ لاین (Pipeline) طراحی کنید و سـپس با زبان توصیف سخت‌افزاری Verilog آن را توصیف کنید. انواع مخاطره ها را تشخیص دهید و برطرف کنید .

برای تست پردازنده خود ابتدا دو برنامه اسمبلی زیر به زبان ماشین تبـدیل کنیـد سـپس آن را در آدرس ۰ حافظـه دستور لود کرده و اجرا کنید. بدیهی است که باید در حافظه داده نیز داده‌های مناسب را قرار دهید تا نتیجه اجـرای برنامه شما مشخص شود.

برنامه ۱: برنامه زیر دو عدد ۱۶ بیتی که در آدرس‌های ۱۰۰ حافظه داده)بایت رتبه پـایین عـدد اول در آدرس ۱۰۰ و بایت رتبه بالای عدد اول در آدرس ۱۰۱(و ۱۰۲ حافظه داده)بایت رتبه پـایین عـدد دوم در آدرس ۱۰۲ و بایـت رتبه بالای عدد دوم در آدرس ۱۰۳(قرار گرفته‌اند را با هم جمع کـرده و حاصـل جمـع ۱۶ بیتـی را در آدرس ۱۰۴ حافظه داده)بایت رتبه پایین حاصل جمع در آدرس ۱۰۴ و بایت رتبه بالای حاصل جمع در آدرس ۱۰۵(قرار مـی-دهد.

```
i ← 0
104[i] = 100[i] + 102[i];         // Add low order bytes
105[i] = 101[i] + 103[i] + ci;    // Add high order bytes with carry
```

```
      LDM   r1, 100(r0)
      LDM   r2, 102(r0)
      ADD   r3, r1, r2
      STM   r3, 104(r0)
      LDM   r1, 101(r0)
      LDM   r2, 103(r0)
      ADDC  r3, r1, r2
      STM   r3, 105(r0)
```

برنامه ۲: برنامه زیر بزرگ‌ترین عنصر را در یک آرایه ۲۰ عنصری با آدرس شروع ۱۰۰ به دست می‌آورد (این مقدار در رجیستر *r7* ذخیره می‌شود).

```
Max = 0;
for (i = 0; i < 20; i++)
      if ( Max < A[i] )
            Max = A[i];
```

```
      ADD   r7, r0, r0
      ADD   r1, r0, r0
Loop: SUBI  r3, r1, 20
      BZ    END
      LDM   r2, 100(r1)
      SUB   r3, r7, r2
      JNC   L1
      ADD   r7, r2, r0
L1:   ADDI  r1, r1, 1
      JMP   Loop
END:
```