



## درس طراحی در سطح سیستم

تکلیف کامپیوتری ۴: پیاده‌سازی پردازنده Leros به کمک Chisel

دانشکده‌گان فنی دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

دکتر بیژن علیزاده

نیم‌سال دوم سال تحصیلی ۱۴۰۳-۰۴

نگارش: امید قلی‌زاده ([omdgzade1380@gmail.com](mailto:omdgzade1380@gmail.com))

### مقدمه

در تکلیف کامپیوتری ۳، با نحوه راه‌اندازی و استفاده از کتابخانه Chisel به کمک محیط توسعه یکپارچه (IDE) IntelliJ IDEA آشنا شدید. در این تمرین می‌خواهیم با به کارگیری ماژول ALU که در تمرین قبلی پیاده‌سازی شد، سخت افزار پردازنده Leros را بصورت سطح بالا توصیف کنیم.

### ساختار پردازنده Leros

این پردازنده که مبتنی بر معماری انباشتگری<sup>۱</sup> است، در دهه‌های ۵۰ و ۶۰ میلادی بسیار استفاده می‌شد. در این پردازنده‌ها، یک رجیستر انباشتگر<sup>۲</sup> وجود دارد که خروجی عملیات پردازشی را در خود ذخیره می‌نماید. بنابراین، در دستورات این پردازنده، ورودی اول همواره مقدار رجیستر انباشتگر است و فقط باید ورودی دوم واحد ALU و نوع عملیات مشخص شود.

ISA<sup>۳</sup> این پردازنده، دستورات با تعداد بیت ثابت ۱۶ بیت و داده‌های ۱۶، ۳۲ و ۶۴ بیتی را پشتیبانی می‌کند. در این تمرین، قصد پیاده‌سازی نسخه ۱۶ بیتی این پردازنده را داریم.

با توجه به دستورات این پردازنده، ۸ بیت بالایی آن (MSB) تعیین‌کننده Opcode، و ۸ بیت پایینی آن (LSB) آدرس خانه رجیستر فایل یا مقدار فوری<sup>۴</sup> است (مطابق جدول ۱).

جدول ۱- قالب دستورات پردازنده Leros

Instruction	Rn/Imm
$b_{15}b_{14}b_{13}b_{12}b_{11}b_{10}b_9b_8$	$b_7b_6b_5b_4b_3b_2b_1b_0$

<sup>1</sup> Accumulator-based

<sup>2</sup> Accumulator

<sup>3</sup> Instruction Set Architecture

<sup>4</sup> Immediate Value



این پردازنده دارای یک رجیستر فایل با اندازه ۲۵۶ است و همچنین حاوی یک رجیستر مخصوص به نام Address Register یا به اختصار (AR) است که آدرس پایه خانه‌ای از رجیستر که می‌خواهد خوانده شود، در آن ذخیره می‌شود. لازم به ذکر است که در دستورات فوری، مقدار ۸ بیت فوری، Sign Extend می‌شوند. لیست دستوراتی که در این تمرین قصد پیاده‌سازی آنها را داریم در جدول ۲ ذکر شده است.

جدول ۲- دستورات مورد نیاز پردازنده Leros

Opcode	Instruction	Function
ADD	000010x0	$A = A + R_n$
ADDI	000010x1	$A = A + Imm$
SUB	000011x0	$A = A - R_n$
SUBI	000011x1	$A = A - Imm$
SHR	00010xxx	$A = A \gg \gg 1$
AND	00100010	$A = A \text{ AND } R_n$
ANDI	00100011	$A = A \text{ AND } Imm$
OR	00100100	$A = A \text{ OR } R_n$
ORI	00100101	$A = A \text{ OR } Imm$
XOR	00100110	$A = A \text{ XOR } R_n$
XORI	00100111	$A = A \text{ XOR } Imm$
Ld	00100000	$A = R_n$
LdI	00100001	$A = Imm$
Str	00110xxx	$R_n = A$
JAL	01000xxx	$PC = A, R_n = PC + 2$
LdAddr	01010xxx	$AR = A$
LdInd	01100x00	$A = MEM[AR + (Imm \ll 1)]$
StrInd	01110x00	$MEM[AR + (Imm \ll 1)] = A$
Br	1000xxxx	$PC = PC + Imm$
BrZ	1001xxxx	If $A == 0$ , $PC = PC + Imm$

برای مثال، دستور "00001001\_00000010" مقدار فوری ۲ را به رجیستر انباشتگر اضافه کرده و نتیجه را در همان رجیستر ذخیره می‌کند. دستور "00001000\_00000011" مقدار ذخیره شده در خانه چهارم ( $R_3$ ) رجیستر فایل را به مقدار رجیستر انباشتگر اضافه کرده و نتیجه را در رجیستر انباشتگر ذخیره می‌کند.

یکی از ویژگی‌های این پردازنده این است که دستورات پردازنده ممکن است از هر ۸ بیت opcode استفاده نکنند. در واقع صفر یا یک بودن بیت‌های Don't Care که با نماد "x" مشخص شده‌اند تاثیری بر انتخاب واحد کنترل ندارند. برای مثال، "10000000" و "10001111" هر دو دستور Br را انجام می‌دهند.

<sup>7</sup> TestBench



#### نکات پایانی

۱. پوشه src، فایل build.sbt، خروجی vcd. و فایل pdf گزارش کار خود را در یک فایل زیپ شده به نام ESL\_CA4\_<StudentId>.zip ارسال نمایید که در آن <StudentId> همان شماره دانشجویی شماست.
۲. در فایل Build.sbt، متغیر name را به صورت CA4\_<StudentId> مقدار داده که در آن <StudentId> همان شماره دانشجویی شماست.
۳. تمامی کد ها باید خوانا بوده و کامنت گذاری مناسب داشته باشد.
۴. در صورت داشتن هرگونه ابهام یا سوال، می توانید از طریق ایمیل (omdgzade1380@gmail.com) و یا تلگرام (@OMD\_Gzade) با من در ارتباط باشید.

#### بارمبندی سوالات

- بخش اول: ۶۵ نمره
- سؤال دوم: ۳۵ نمره
- تمیزی گزارش و مرتب بودن فایل های آپلود شده نمره امتیازی دارد.

موفق و سلامت باشید.